Improving Autonomous Robot Photography using RGB-D Sensors

Manfredas Zabarauskas

Keble College

University of Oxford

Supervised by: Dr Stephen Cameron

Submitted in partial fulfilment of the degree of

Master of Science in Computer Science

September 5, 2013

Abstract

Autonomous robot photographers serve as excellent low-cost robotics research platforms, encompassing difficult multidisciplinary challenges. For their successful operation, they need to *i*) detect and track people in their environment, *ii*) autonomously wander around avoiding active and passive obstacles, *iii*) compose aesthetically pleasing pictures, *iv*) intelligently interact with humans, and so on.

All of the robot photographer systems described in scientific literature between 2003–2013 (since the earliest robot photographer, to the present day) rely on RGB cameras for their vision, and laser range/infrared/ultrasound sensors for obstacle detection and avoidance. The work presented in this thesis describes how the combined colour/depth data provided by affordable and ubiquitous RGB-D sensors can be used to improve the current state-of-the-art in autonomous robot photography.

To that end, this thesis thoroughly surveys previous solutions to the main challenges of robot photographers (*viz.* human subject detection/tracking and obstacle detection/avoidance) and proposes a number of novel methods or existing method extensions to solve these problems. It also describes solutions to other robot photographer's tasks, like photograph composition, RGB-D and photographic camera alignment, or robot's state externalization.

After the theoretical description of the methods, their implementations within an open-source Robot Operating System framework are described. To evaluate this software in real-world situations, a physical robot containing a point-and-shoot photographic camera, a low-cost RGB-D Microsoft Kinect sensor and an open-source hardware platform is built and deployed in an unstructured real-world event[1]. More than half of the pictures taken by the robot in this event are evaluated by independent judges as "good (4)" or "very good (5)" (on a five-point Likert scale), markedly exceeding the best available previous results reported by Byers et al. (2003) and Ahn et al. (2006).

[1]A video of the robot "in-action" can be seen at `http://zabarauskas.com/robot-photographer`.

Contents

List of Figures

List of Tables

Chapter 1

Robot Photographers in the Context of Autonomous Robots

This chapter presents a brief history of robots and provides examples of modern autonomous robots in a variety of application areas. Within this context, it describes the motivation for autonomous robot photographer research and explains the applicability of low-cost RGB-D sensors in this task. At the end of the chapter, the project's aims and success criteria are presented, and the structure for the rest of the dissertation is explained.

1.1 Brief history of autonomous robots

The idea of artificial creatures exhibiting human-like behaviour has been around for centuries:

- Homer's *Iliad* (760-710 BC) describes mechanical tripod handmaids made from gold by Hephaestus, the Greek god of fire and crafts (Monro, 1903),
- an ancient Chinese text *Liezi* (列子, 400-300 BC) tells a story of the craftsman Yanshi, who created a life-sized humanoid (indistinguishable from a human in looks and behaviour) to impress the King Mu of Zhou (Needham, 1991),
- Apollonius' *Argonautica* ($\sim$300 BC) mentions Talos, a giant man-like creature made out of bronze (with lead in its veins), potentially created by Daedalus, the skilful craftsman in Greek mythology (Peris, 1971).

However, the first real-world materializations of autonomous robots appeared only in late 20th century. One of the earliest autonomous robots described in the scientific literature was Shakey, built around 1970 in SRI (Nilsson, 1984). Shakey could perform basic tasks given by an operator within a highly-controlled coloured block world. Nine years later, another robot from SRI called the "Stanford Cart" (Moravec, 1980) managed to autonomously cross a chair-filled room (albeit in five hours). Both of these robots used TV cameras for their vision.

During the three subsequent decades robotics research has spurred, producing walking, swimming, driving and flying; anthropomorphic[1] and non-biomimetic[2] autonomous robots. These robots used various types of vision/distance sensors, and have been applied in a number of areas, ranging from space exploration to entertainment. Some examples of the modern autonomous robots and the sensors which they used for image and depth inputs are presented in tables 1.1, 1.2 and figures 1.1, 1.2.

1.2 Motivation for "robot photographer" research

Within the field of autonomous robotics (and the variety of its application areas), robot photographers serve as excellent low-cost research platforms. They encompass a number of challenges common in robotics research, like task and path planning, locomotion and navigation (including obstacle avoidance), and human subject detection/tracking.

Robot photographers also include multidisciplinary challenges, like the automatic photograph composition (which requires computational understanding of the aesthetics of photography) and Human-Robot Interaction (HRI). As

[1]Human/animal-like.
[2]Machine-like.

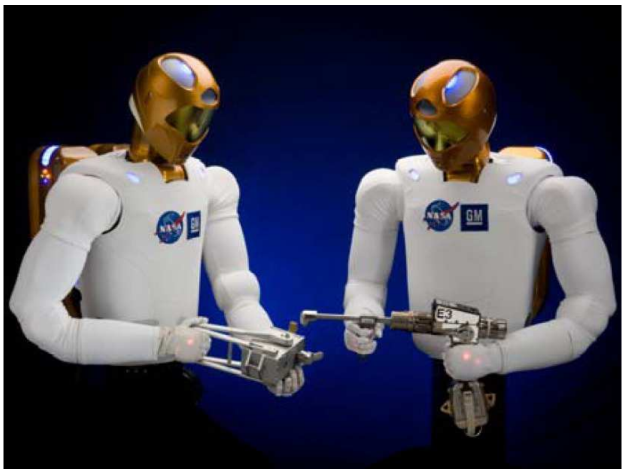
Area	Application	Example autonomous robot	Depth sensors	Image sensors
Military	Urban search and rescue (USAR)	Hector, an unmanned ground vehicle for autonomous victim search and environment mapping (Graber et al., 2013)	Laser range scanner, RGB-D sensor	RGB camera
	Humanitarian demining	Gryphon, an autonomous landmine detection robot (Fukushima et al., 2008)	Stereo RGB camera pair	Stereo RGB camera pair
	Reconnaissance	Autonomous unmanned ground and aerial vehicles for military threat detection (Janko et al., 2011)	Laser range scanners, ultrasonic sensors	RGB cameras
	Surveillance	Security robot for autonomous ship cabin monitoring (Chung, 2013)	Laser range scanners, ultrasonic sensors	RGB camera
Space exploration	Planetary surface exploration	Curiosity, a car-sized Mars rover (Grotzinger et al., 2012)	Six monochrome stereo camera pairs	Monochrome telescopic camera, two RGB mast cameras, robotic arm RGB camera, descent RGB camera
	Astronaut assistance	Robonaut 2, an anthropomorphic robot helper, deployed in International Space Station (Diftler et al., 2011)	Stereo camera pair, infrared depth sensor	Two RGB cameras
	Landing on airless planetary bodies	WGTA, an autonomous lander flight test vehicle (Chavers et al., 2012)	Inertial measurement unit, altimeters	High-resolution RGB camera
Industry	Manufacturing	Baxter, an adaptive manufacturing robot, trainable by non-technical personnel (Fitzgerald, 2013)	Ultrasonic sensors	Three RGB cameras
		Little Helper, an autonomous manufacturing assistant robot (Hvilshoj et al., 2009)	Laser range scanner, ultrasonic sensors	Monochrome camera
	Welding	Self-correcting gas tungsten arc welding robot (Shen et al., 2010)	-	RGB camera
	Harvesting	Apple harvesting robot (De-An et al., 2011)	Infrared sensors	RGB camera
	Logistics	Unmanned forklift for palletized cargo handling (Teller et al., 2010)	Laser range sensors	RGB cameras

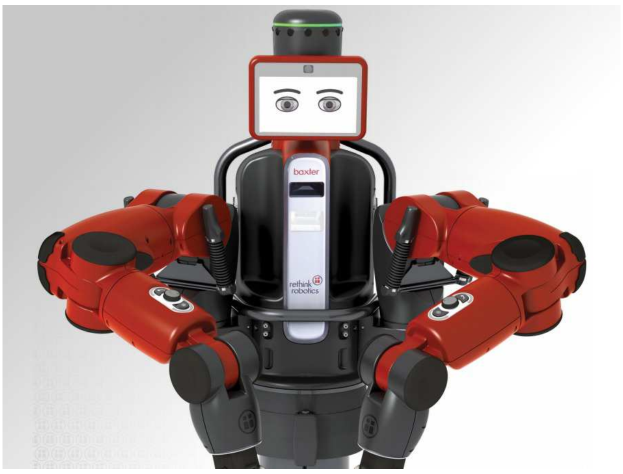
Table 1.1: Examples of state-of-the-art robots with varying degrees of autonomy in different application areas, together with their depth/image sensors.

Area	Application	Example autonomous robot	Depth sensors	Image sensors
Assistive robotics	Assistance for elderly	Care-O-bot 3, a mobile robot butler (Graf et al., 2009)	Stereo RGB camera pair, Time-of-flight camera, laser range scanner	Stereo RGB camera pair
	Assistance for blind	eyeDog, an assistive guide robot (Galatas et al., 2011)	Laser range sensor	RGB camera
	Assistance for physically disabled	Caregiver-following robotic wheelchair (Wu et al., 2013)	RGB-D sensor	RGB-D sensor
	Autism research	Bandit, a humanoid bubble-blowing robot (Feil-Seifer and Matarić, 2008)	Laser range scanner	RGB cameras
	Domestic services	Roomba, a robotic vacuum cleaner (Forlizzi and Disalvo, 2006)	Infrared sensors, cliff/wheel-drop/bumper sensors	-
	Childcare	PaPeRo, a childcare robot (Osada et al., 2006)	Stereo RGB camera pair, ultrasonic sensors	Stereo RGB camera pair
Entertainment	Tour guiding	Neptuno, an autonomous tour guide robot (Bueno et al., 2011)	Laser range sensor, ultra-wide band (UWB) sensors	-
		Robotinho, a humanoid tour guide robot (Faber et al., 2009)	Laser range sensor, ultrasonic distance sensors	Two RGB cameras
	Robotic football	A robot goalkeeper (Dias et al., 2013)	RGB-D camera	Two RGB cameras
		Turtle, a football playing robot (Hoogendijk et al., 2012)	Laser range scanner	Two RGB cameras
	Robotic receptionists	Hala, a bilingual robot receptionist (Simmons et al., 2011)	Laser range scanner	-
		Olivia, a child robot receptionist (Niculescu et al., 2010)	Stereo RGB camera pair, laser range scanner	RGB camera
	Robotic pets	Pleo, a robot pet dinosaur (Gomes et al., 2011)	Four ground sensors	RGB camera
	Robot photographers	Human-interaction based robot photographer (Ahn et al., 2006)	Ultrasonic and infrared sensors	Two RGB cameras
		Lewis, an autonomous event photographer (Byers et al., 2003)	Laser range sensor	Two RGB cameras

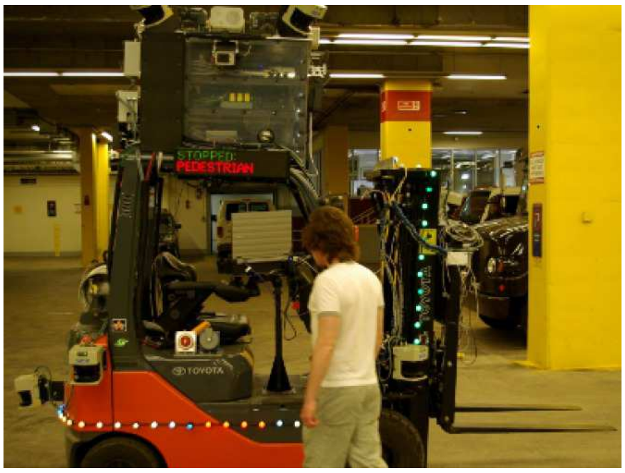
Table 1.2: Continued examples of state-of-the-art robots with varying degrees of autonomy in different application areas, together with their depth/image sensors.

Hector, an unmanned ground vehicle for autonomous victim search and environment mapping. Taken from Graber et al. (2013).


Robonaut 2, an anthropomorphic robot helper, deployed in International Space Station. Taken from Diftler et al. (2011).


Baxter, an adaptive manufacturing robot, trainable by non-technical personnel. Taken from Rethink Robotics (2013).

An autonomous apple harvesting robot. Taken from De-An et al. (2011).



Unmanned forklift for palletized cargo handling. Taken from Teller et al. (2010).

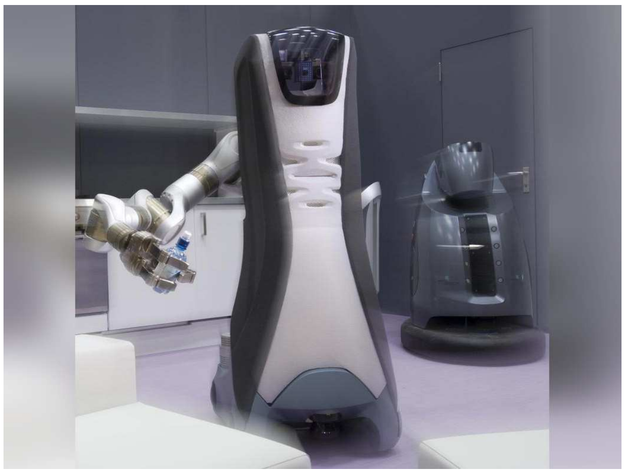
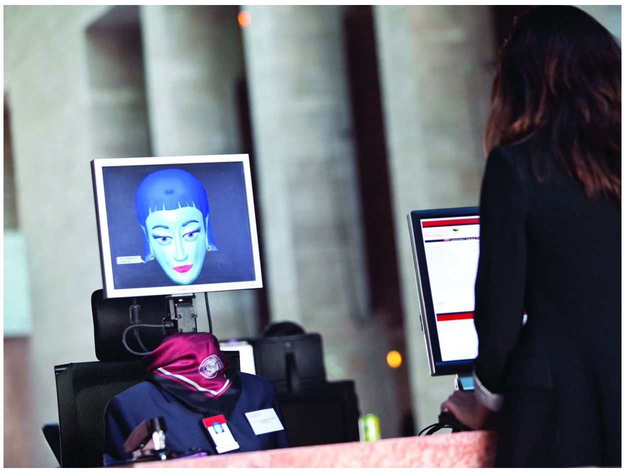

Care-O-bot 3, a mobile robot butler. Taken from Graf et al. (2009).

Figure 1.1: Examples of state-of-the-art robots with varying degrees of autonomy.

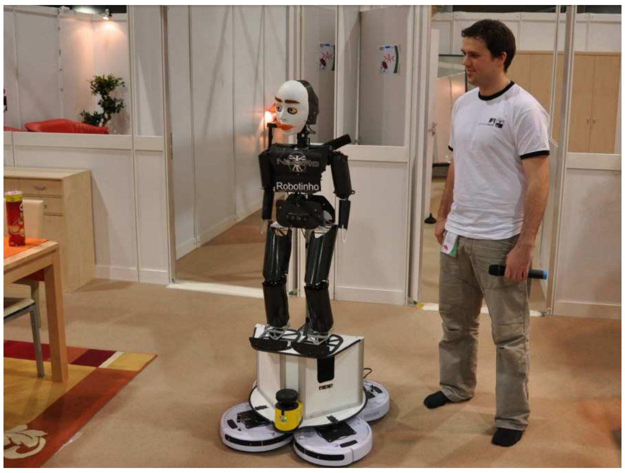
Bandit, a humanoid bubble-blowing robot. Taken from Feil-Seifer and Matarić (2008).



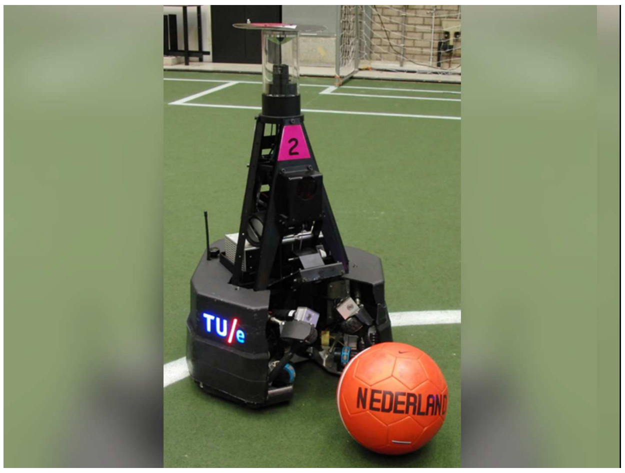
Hala, a bilingual robot receptionist. Taken from Simmons et al. (2011).

PaPeRo, a childcare robot. Taken from Osada et al. (2006).

Pleo, a robot pet dinosaur. Taken from Innvo Labs (2013).



Robotinho, a humanoid tour guide robot. Taken from Faber et al. (2009).



Turtle, a football playing robot. Taken from Hoogendijk et al. (2012).

Figure 1.2: Continued examples of state-of-the-art robots with varying degrees of autonomy.

pointed out by Byers et al. (2003), robotic photographer platforms are particularly well suited for HRI research, since the general public can easily grasp the overall concept ("it's a robot whose job is to take pictures"), and thus tend to interact with the robot naturally.

Autonomous robot photographers are also interesting as undergraduate teaching tools. In particular, they could be used as hands-on research platforms in machine learning/artificial intelligence and similar courses. They could also be used for engaging prospective Computer Science undergraduates during the "open day"-like events.

Finally, robot photographers show potential in commercial applications (*viz.* event photography, robotic journalism or workplace monitoring), since the service costs of an autonomous robot photographer are significantly smaller than those of a professional photographer.

1.3 Low-cost RGB-D sensor applicability in autonomous robotic photography

As indicated in tables 1.1 and 1.2, autonomous robots often need depth and colour data to successfully interact with their environment. While various types of cameras can easily provide the colour data, a number of different sensors have been investigated for acquiring the distances to objects in the robot's surroundings. These sensors include laser range scanners, monocular/stereo camera rigs and infrared/ultrasonic sensors. Compared to the active-light RGB-D sensors, they all have their own disadvantages (as summarized in table 3.1).

The low-cost RGB-D sensors like Microsoft Kinect or Asus Xtion Pro contain RGB and infrared (IR) cameras, and an IR projector which emits a structured IR pattern. The structured-light triangulation based on the input from the IR camera is then used to infer object depths in the scene. This approach yields dense three-dimensional point clouds at around 30 frames per second, while consuming smaller amount of power then LIDARs and providing more accurate data than ultrasound sensors or stereo camera pairs (*e.g.* the latter need to solve the correspondence problem, which is problematic in homogeneous color areas).

These features of RGB-D sensors and their successful adoption in a number of other recent autonomous mobile robots (*e.g.* see the robots by Stückler and Steffens (2011), Graber et al. (2013), Wu et al. (2013)) suggest that they could be used to advance the current state-of-the-art in autonomous robot photography.

To that end, this dissertation focuses on investigating RGB-D data based solutions for human detection/tracking, obstacle avoidance and other challenges of autonomous robot photographers. The final choice of the algorithms is strongly influenced by their computationally simplicity, since it directly translates to energy efficiency (longer battery life) and their potential to be executed on more modest configuration laptops/netbooks, making the autonomous robotic photography research more accessible both within and outside of the academia.

1.4 Project aims and success criteria

With the above motivation in mind, the main aims of the project are as follows:

1. Investigate alternative solutions (based on RGB-D data) to the main problems in robot photography, focusing on:
    (a) human subject detection/tracking and obstacle avoidance tasks,
    (b) processing power, and the associated energy usage constraints imposed by mobile robotics.

2. Evaluate the real-world effectiveness of these solutions by implementing them in an autonomous "event photographer" robot. This includes:
   (a) creating a structural design, obtaining the hardware components and assembling the physical robot,
   (b) creating a modular software design and implementing the proposed algorithms,
   (c) deploying the robot photographer in an unstructured environment (e.g. an open-day event) and comparing the empirical results with those obtained by Byers et al. (2003) and Ahn et al. (2006).
3. Contribute to increasing the accessibility of autonomous robot photography research both within and outside academia by:
   (a) using widely available, low-cost off-the-shelf hardware components for the robot's structural design,
   (b) designing modular, high-cohesion and low-coupling robot control software (to promote investigation into robot's performance changes after replacing a particular software component),
   (c) open-sourcing the hardware and software design, and the source code of the implemented autonomous robot photographer.

The project will be considered to be successful if the following goals are achieved:

1. The hardware and software implementations of the robot are completed within the four-month timeframe of the project,
2. The robot is capable to autonomously navigate and take well-composed photographs of human subjects when deployed to an unstructured environment,
3. The quality of the photographs (when evaluated on a five-point Likert scale) approaches the results reported by Byers et al. (2003) and Ahn et al. (2006).

1.5 Dissertation structure

This thesis is structured in a chronological order, following different stages of an autonomous robot photographer's development (*viz.* analysis, implementation, deployment, evaluation and *post-mortem* learning):

- *"Chapter 2: Advances and Limitations of Robotic Photography Research"* introduces the research done in autonomous robot photography between 2003–2013 (from the first autonomous photographer robot described by Byers et al. (2003), to the present day). At the end of the chapter, the limitations of the earlier approaches are summarized.
- *"Chapter 3: Solving Main Robot Photographer Challenges with RGB-D Data"* describes two main problems in robotic photography that can be solved using RGB-D data: human subject detection/tracking and obstacle detection/avoidance. It then presents an in depth survey of the existing methods for solving these problems based on RGB and depth data (separately and combined). After the survey, the proposed methods for solving these problems in the context of an autonomous robotic photography are described.
- *"Chapter 4: Development of "Luke": an Event Photographer Robot"* presents the implementation of an autonomous robot photographer. It provides the details of hardware components, software architecture and individual module implementations, as well as their run-time performance measurements.

- *"Chapter 5: Insights from Robot Photographer's Deployment in Real-World"* summarizes the experiences from Luke's deployment in an unstructured real-world event and provides statistical evaluation of the taken pictures. The obtained results are compared with the ones reported for earlier robot photographer approaches.
- *"Chapter 6: Conclusions and Proposals for Further Research"* sums up the work completed in this dissertation and the contributions to the field of autonomous robot photography. It ends the dissertation with the discussion of potential directions for future research.

Chapter 2

Advances and Limitations of Robotic Photography Research

This chapter summarizes the research of autonomous robot photography, starting with the seminal work of Byers et al. (2003) and continuing with the subsequent methods proposed the last decade. At the end of the chapter, the limitations of previous approaches are summarized.

2.1 Survey of existing robot photographers

Autonomous robot photographers have first been developed in 2003 (rather late in the context of the autonomous robots). Due to the the amount of challenges that autonomous robot photographers need to solve for the successful operation, only a handful of photographer robots have been described up to this date, despite the benefits described in section 1.2. All of the autonomous photographer robots described in scientific literature between 2003–2013[1] are summarized below.

2.1.1 Lewis, the first autonomous "event photographer" robot (Byers et al., 2003)

The earliest implementation of an end-to-end mobile robot system capable of taking well-composed pictures is described by Byers et al. (2003, 2004), Smart et al. (2003). Their mobile robot, Lewis, is equipped with an on-board laptop (Intel Pentium III 800 MHz CPU, 128 MB of RAM), a laser range finder and two cameras: a VGA (640×480 pixel) resolution, 30 frames-per-second (FPS) video camera, and a 1.9 mega-pixel (MP) photographic camera. In order to find the human subjects in the environment, Lewis starts with human skin detection in VGA camera's video feed using an approach inspired by Forsyth and Fleck (1999). This approach is based on the insight that human skin occupies an easy-to-define region in a color space.

For every new environment that Lewis is deployed in, a human annotator needs to identify skin pixels in a small number of initial training images to increase skin detector's invariance to lighting conditions. Using the "skin"/"non-skin" color space areas (derived from these annotations), every pixel in an input VGA frame is classified as belonging (or not) to a skin region, producing a binary "skin map" image. The contiguous regions in the skin map are considered as candidate faces.

Using simple geometry, Lewis associates the detected candidate face patches with readings from a laser range finder (which produces 180 radial distance measurements in front of the robot). After making a further assumption that all human subjects in the environment are standing adults, the candidate face patches which do not fall into pre-defined ranges of face sizes and distances from the ground are discarded.

Lewis has two navigation modes: random and intentional. In the random mode, the robot simply avoids the obstacles while looking for suitable photographic opportunities. In the intentional navigation mode, Lewis pro-actively navigates to a location that maximizes the photo quality objective function which considers the distance from subjects, reachability, or presence of obstacles in the path, inter-subject occlusion, *etc.* Byers et al. note that the random navigation mode works best in crowded environments.

After reaching the suitable photographic location, Lewis attempts to achieve a pleasing photographic composition (based on simple composition rules, like a "rule-of-thirds") by panning, tilting and zooming the 1.9 MP camera in

[1]To the best of author's knowledge.

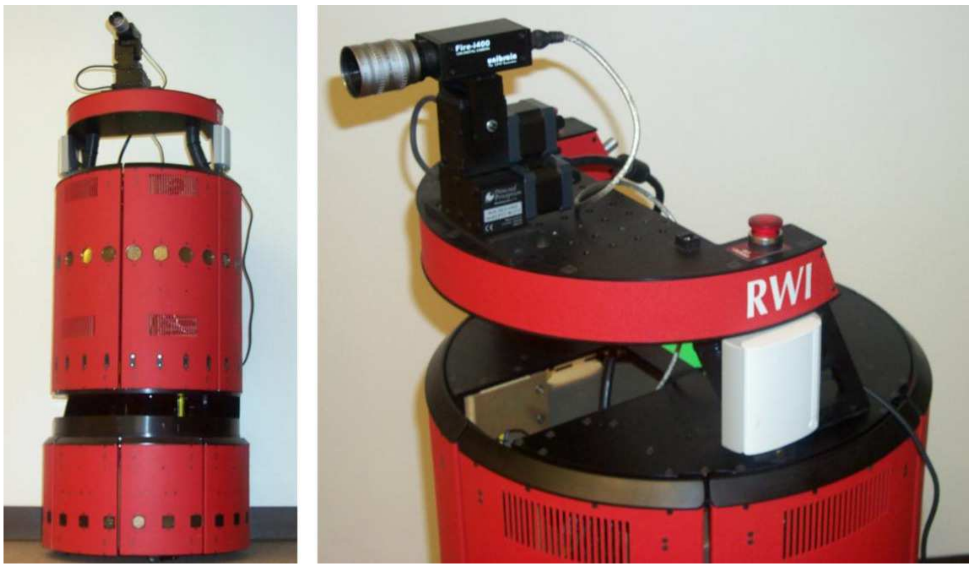


Figure 2.1: Lewis, an autonomous "event photographer". Taken from Smart et al. (2003).

a closed-loop fashion. When the picture is taken, it is automatically transmitted to a "viewing station", where the attendees of the event can browse, print or e-mail the photographs.

2.1.2 Table-top robot capable of human/non-human group picture framing (Campbell and Pillai, 2005)

Campbell and Pillai (2005) describe a limited-mobility robot system (moving on a table surface), capable of taking well-framed group pictures of human and non-human subjects in static environments. All visual computations are performed using an adjacent PC with Intel Xeon 3.6 GHz CPU, and two RGB cameras connected to their robot: a high-resolution (5 MP) photographic camera, and a 10 FPS VGA webcam, attached to a pan-tilt head. The high-resolution camera also operates in a "viewfinder" mode, yielding 5 FPS at QVGA (320×240 pixel) resolution. In order to simplify the hardware design and to avoid camera calibration/alignment problems, subject detection is performed using the same photographic camera operating in the "viewfinder" mode.

To detect subjects, the robot relies on the assumptions that the scene is static, the photographic subjects are grouped together (in a plane parallel to the photographic camera's image plane) and are clearly separated from the background depth-wise. To identify the subjects closest to the camera, the robot translates 15 centimeters in a direction parallel to the camera's image plane and measures the amount of optical flow in the scene using the method by Lucas and Kanade (1981). The optical flow field vectors are then clustered based on their velocity, and inconsistent flow vectors for the same scene region are eliminated to reduce tracking error noise and ignore minor movements in the scene.

Campbell and Pillai defines the region-of-interest (ROI) of a scene as the closest sufficiently large flow vector cluster, where distances to the clusters are obtained based on the observation that the amount of motion parallax (and thus the correlated optical flow field velocity) is inversely proportional to the static scene's depth. After obtaining the subject ROI, the photograph is taken in a high-resolution mode, the ROI is cropped out from the original image and, if necessary, scaled/cropped from the bottom to fit the 3:2 output aspect ratio. This allows for the cropped out photograph to be printed as a 4×6 inch landscape image on a connected color inkjet printer.

Given that the robot itself moves on a table surface, a VGA webcam is used for hazard detection. Before moving the robot, this camera is panned/tilted to face the direction of travel. The robot is then moved by one centimeter

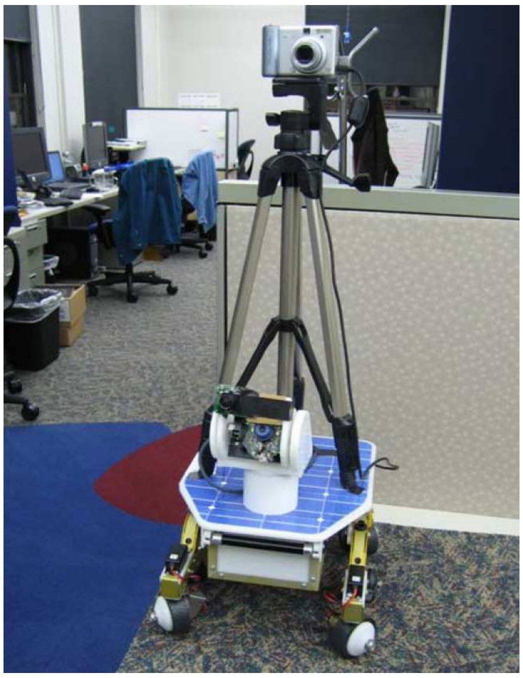


Figure 2.2: Table-top robot capable of group picture framing using optical flow. Taken from Campbell and Pillai (2005).

and the amount of motion parallax within the scene is estimated using the same optical flow method. A consistent presence of a large discontinuity in the optical flow field is treated as an indication of a table edge, in which case the robot moves to an opposite direction (to permit the required 15 centimeter translational movement).

2.1.3 Robot photographer capable of basic interaction with humans (Ahn et al., 2006)

Ahn et al. (2006) focus on the human-robot interaction aspect of autonomous robot photographers. They describe an interactive robot photographer with limited mobility, which supports two modes of operation: "background priority" and "profile shot". In the former mode, the robot attempts to maximize the amount of background visible in the resulting shot by moving closer to/away from the subjects as required for optimal framing; in the latter mode, the robot takes the profile shot of a subject with the zoomed-in camera. In both modes, simple photographic composition heuristics (like the "rule-of-thirds", or "no-middle" rule) are applied to obtain the framing objective.

The choice between "background priority" or "profile shot" photographic modes is made by human subjects, (somewhat unintuitively) by waving the left or right hand. In order to detect human subjects and waving gestures, the robot uses an input from a VGA resolution RGB camera (processed using an on-board PC with a Pentium IV 3 GHz CPU and 1 GB of RAM, all integrated into an ETRO mobile robot).

For face detection, a *de facto* industry standard Viola and Jones (2001) object detection framework is used. In this approach, a cascade of "strong" classifiers is composed from Haar-like wavelet features ("weak" classifiers), using a variant of AdaBoost (Freund and Schapire, 1997) classifier boosting procedure (more details are given in section 3.1). To reduce the computational complexity of human subject detection, the robot tracks detected faces until they are lost using the mean-shift based object tracker applied to color-histogram backprojected images (Agbinya and Rees, 1999).

If any faces are detected/tracked, the robot looks for waving gestures in the image regions where the subject's hands should be, using heuristics about the human body shape. If these regions contain movement (obtained by frame

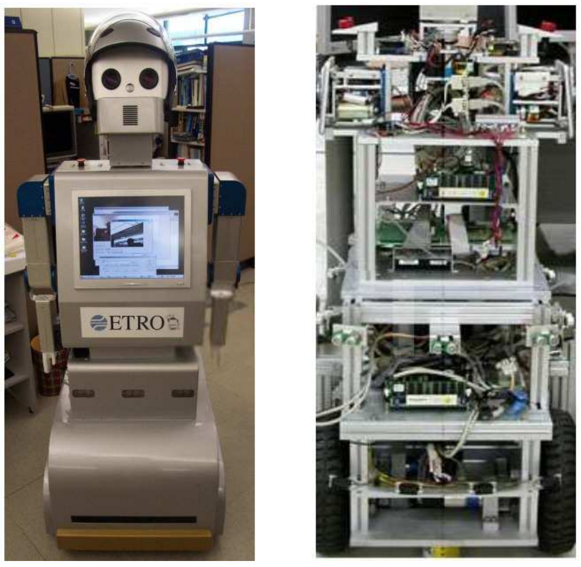


Figure 2.3: ETRO robot platform, on which Ahn et al.'s (2006) robot photographer is based. Left image is the exterior, right image is the interior of ETRO's platform. Taken from Ahn et al. (2006).

difference) and skin-colored pixels for a number of consecutive frames, a waving gesture is registered.

The same VGA camera is used for taking the final pictures, which allows Ahn et al. to avoid multiple camera alignment/calibration problems. However, since this camera does not have a built-in flash, resulting images have poor quality in difficult lighting conditions. To mitigate this problem, the robot optionally applies the "retinex" localized color constancy algorithm (Land and McCann, 1971), in an attempt to retrieve the real surface reflectance by discounting the illuminant light. The choice on whether retinex picture enhancement algorithm should be applied is exposed to the user through the LCD panel attached to the chest of the robot (which also displays the resulting picture).

2.1.4 Robot photographer capable of human localization by sound direction (Kim et al., 2010)

Kim et al. (2010) propose a mobile photographer robot, which can direct its attention to the human subjects using audio input. In particular, the robot uses three analogue microphones arranged in a circle at 120° intervals and the time-difference-of-arrival (TDOA) method to localize the sound source using a FPGA (Jin et al., 2008). After localizing the sound source, the robot rotates towards it and starts searching for the human subjects in the RGB input image. To that end, input from an on-board VGA camera is searched for contiguous patches of human skin.

For "skin"/"non-skin" pixel classification, a Mixtures of Gaussian model (with four Gaussian components) is fitted to the skin pixel distribution of the ground-truth tagged training database. Then a given pixel is classified as belonging to the skin if its Mahalanobis distance to the nearest cluster centroid is within a pre-set threshold (Pham et al., 2009). The top-most sufficiently large and dense skin component in the input image is assumed to correspond to the human subject's head.

After detecting the human subject, the robot rotates and translates as required to satisfy the "rule-of-thirds" photographic composition heuristic. In an attempt to further improve the composition quality, the robot tries to avoid the unintentional dissection lines (lines in the picture that are cutting through the subjects body, see Shen et al. (2009) and figure 2.4).

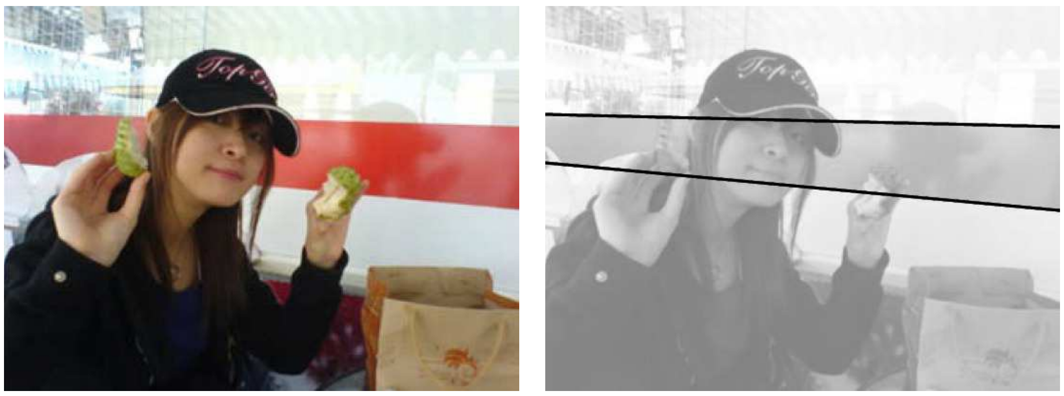


Figure 2.4: An example of unintentional dissection lines (black lines in the right image). The red stripe in the background of the original image (on the left) crosses the subject's head and thus distracts the visual attention from the subject. Taken from Shen et al. (2009).

To detect such lines a Canny edge detector (Canny, 1986) is combined with the line detection by Hough transform (Duda and Hart, 1972). If no unintentional dissection lines are found and the human subject is positioned close-enough between the one-third and two-thirds horizontal lines, Kim et al.'s robot takes the picture using the same VGA camera. All visual computations of the robot are performed using an on-board CPU with an Athlon 64 X2 Dual Core 2.01 GHz CPU with 2.5 GB RAM.

2.1.5 Aesthetic guideline driven robot photographer for static scenes (Gadde and Karlapalem, 2011)

Gadde and Karlapalem (2011) present a stationary robot based on a humanoid NAO platform (Gouaillier et al., 2008), which can take pictures of static scenes containing both human and non-human subjects. Their robot works in an iterative fashion: first of all, a picture is taken using a HD resolution (1.2 MP) camera, mounted on the robot's head. Then the picture's quality is evaluated using three aesthetic criteria explained below. If picture's quality is under a pre-set threshold, the camera is repositioned by changing robot head's angle, a picture is re-taken at a new position and the process is repeated until the required quality threshold is reached.

Three types of metrics are used to evaluate the aesthetic appeal of a picture: i) the position of the photograph's focus region (extracted using a visual saliency model described below) with respect to the "rule-of-thirds", ii) the position of the horizon line with respect to the "golden-ratio" rule, and iii) the "high-level" visual features of the picture (*viz.* clarity contrast, lighting, simplicity and color harmony).

To extract the region of focus in a given image, the visual attention model by Achanta et al. (2009) is used. In this model, the salience map S is obtained by calculating $S(x, y) = |I_\mu - I_b(x, y)|$, where I_μ is the mean pixel value in input image and I_b is the input image blurred using a Gaussian kernel. The original image is then segmented using the mean-shift method, and the segments with an average saliency value larger than an adaptive threshold (two times the mean saliency of the image) are considered to be the focus regions. The picture's conformance to the "rule-of-thirds" is evaluated by examining the deviation of centroid of the focus region from the ideal locations as indicated by this composition rule.

To extract the position of the horizon line, Gadde and Karlapalem's robot uses the vanishing point detector by Leykin (2006). Essentially this detector works by convolving the input image with *horizontally* oriented Gabor wavelets and retaining the position/angle of the strongest cumulative response (indicating the location of the horizon line

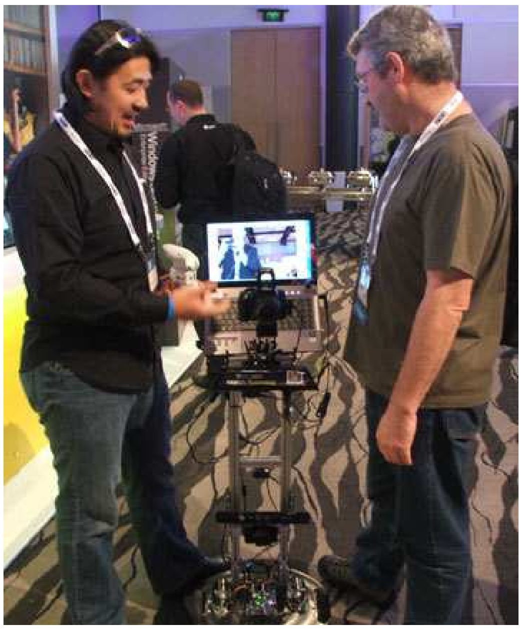


Figure 2.5: Roborazzi II, a party photographer. Taken from Gardner (2012).

in the image). The position of the horizon line is then evaluated *w.r.t.* the "golden-ratio" rule (the ratio between the areas of rectangles formed by dividing the image using the horizon line should be equal to φ).

Finally, to evaluate the image based on the focus region's visual features, Gadde and Karlapalem train a two-class ("good image"/"bad image") Support Vector Machine (SVM) classifier using 12,000 ranked training images. Clarity contrast, lighting, composition and color features features used in training and classification are based on brightness ratio, spectral (Fourier domain), and HSV/RGB histogram properties, as described by Luo and Tang (2008).

2.1.6 Roborazzi II, a party photographer (Shirakyan et al., 2012)

Roborazzi II, a party photographer robot was presented by Shirakyan et al. in Microsoft TechEd 2012 event in New Zealand. It is designed as a showcase application for Microsoft Robotics Developer Studio 4 software suite.

For its *modus operandi*, the robot seems[2] to be using two Microsoft Kinect RGB-D sensors to roam around the room avoiding the obstacles and tracking humans. The sensor used for obstacle avoidance is pointing downwards, while the sensor for human subject detection is facing forwards. If the robot detects a person ahead, it rotates to center the person in its FOV (using some rule(-s) of photographic composition), produces an audio signal ("Say cheese!") and takes a picture with a high-resolution DSLR camera. Afterwards, the picture is automatically uploaded to Flickr. All computation seems to be performed using an on-board PC.

As can be inferred from various (mostly non-technical) presentations about the robot, Roborazzi's vision algorithms work as follows. For the obstacle avoidance, the robot initially calculates a first-order discrete spatial derivative of the depth image from the downwards-facing Kinect sensor (thereby performing a primitive edge detection in the depth image). Then, any points in the image where the derivative exceeds a certain threshold are marked as obstacles, and the robot navigates away from these points while roaming around the room.

For the human detection, Shirakyan et al. use the skeletal tracking API with Kinect for Windows SDK. In this API, skeletal tracking is implemented using the approach by Shotton et al. (2011): in each depth frame individual

[2]Since no scientific/technical descriptions exist of Roborazzi II, only a very limited amount of information can be obtained about the internal workings of the robot.

depth pixels are classified as belonging to one of 31 body parts using three randomized decision trees (each of 20 levels deep), trained using 1 million images on a 1000-core cluster. Then a mean-shift (Comaniciu and Meer, 2002) approach (with a Gaussian kernel) is used to find the modes of the classified body parts, which are returned as proposed skeleton joints.

2.2 Limitations of earlier approaches

In the context of the project aims described in section 1.4, the main deficiencies of earlier robot photographers approaches can be summarized as follows:

- *Unreliable human subject detection*. The robots by Byers et al. (2003) and Kim et al. (2010) use skin pixel detection from the input RGB image as their *main* method of human localization, even though it is well-known to be unreliable in varying illumination conditions (for example, see Kakumanu et al. (2007)). Furthermore, Kim et al.'s approach is limited to a single human-subject detection. Ahn et al. (2006) use the Viola-Jones face detector cascade, which is unable to detect faces with out-of-plane rotations (*e.g.* profile faces), since the cascade is trained using only frontal face data. Roborazzi II by Shirakyan et al. (2012) is limited to tracking at most two human subjects at a time and cannot cope with non-standing human poses (limitations of skeletal tracking API).
- *Unsuitability to dynamic scenes*. Campbell and Pillai's (2005) and Gadde and Karlapalem's (2011) robots are not suitable for environments where the subjects are moving around. In Campbell and Pillai's case movement in the background would confuse the optical flow algorithm, while movement of the subjects in the foreground would confuse the motion parallax estimation and would break the assumption that subjects are in a plane parallel to the camera. In turn, Gadde and Karlapalem's robot relies on the fact that environment remains static while it reorients the camera to improve the picture's composition. If subjects keep moving, the robot might not converge to a composition that reaches the necessary quality threshold.
- *Limited mobility*. Campbell and Pillai's robot moves only on a table surface (the authors argue that it is done to "better interact with humans and to photograph human subjects from a more pleasing angle", Campbell and Pillai (2005)). The robot by Ahn et al. rotates towards the caller and then moves only backwards and forwards to satisfy the "rule-of-thirds" in "background priority" mode (as described above), or zooms the camera towards the caller in "profile shot" mode. Gadde and Karlapalem's robot only reorients the camera angle (which is mounted in a humanoid robot's head) to satisfy the composition quality criteria.
- *Inadequate photographic camera quality*, due to the lack of flash (in robots by Ahn et al., Kim et al. and Gadde and Karlapalem) and low resolution (0.3 MP by Ahn et al. and Kim et al., 1.2 MP by Gadde and Karlapalem, 1.9 MP by Byers et al.).
- *Limited availability for further research/teaching*, since i) the source code/hardware designs are not publicly available for any of the robots described in this chapter, which significantly restricts reproducibility of the results and further research, ii) the robots use expensive equipment (like the laser range finder used by the robot of Byers et al.) or are built on expensive platforms (like NAO, used by Gadde and Karlapalem), and iii) most of the platforms on which the robots are based are proprietary (*i.e.* not available off-the-shelf; this includes robots by Byers et al., Campbell and Pillai, Ahn et al. and Kim et al.).

The autonomous event photographer robot proposed in the following chapters of this dissertation attempts to lift these limitations.

Chapter 3

Solving Main Robot Photographer Challenges with RGB-D Data

This chapter surveys the solutions to two main problems in autonomous robotic photography: human subject detection/-tracking and obstacle detection/avoidance. A reliable solution to the human detection/tracking problem is essential since the quality of a photo's composition depends directly on accurate human localization in the image. Likewise, a solution to the obstacle detection and avoidance problem is necessary for the robot to be able to randomly wander about in an unstructured environment, while avoiding collisions with any obstacles that may appear in its path.

This chapter starts by thoroughly surveying previous approaches to solving these challenges using colour and depth data (separately or combined). A reader familiar with the typical computer vision/machine learning solutions to these tasks is encouraged to skip directly to the end of the chapter (section 3.3), where the proposed RGB-D data based methods are described.

3.1 Human subject detection

As discussed in the project's aims section (1.4), the robot should be able to produce well-composed pictures of humans during various events. To accomplish this task, an autonomous robot photographer needs to be able to detect human subjects in its environment.

Since this task is absolutely crucial to the robot photographer's performance (inaccurate human subject detection leads to wrong picture composition, which in turn leads to bad quality pictures), the sections below summarize these techniques in detail for both image and depth data, starting with human detection in images.

3.1.1 Survey of image-based body/face detection methods

The goal of human body/face detection tasks is to determine the positions and sizes of human bodies and/or faces if they are present in the input images. The human body detection task is complicated due to a large within-class variability of human images, arising from changes in pose or clothing, lighting, occlusions and background variations. The face detection task is further complicated by the additional sources of variance arising from presence/absence of facial features (beards, moustaches, different haircut styles, glasses), and changes in facial expression.

Due to their immediate applicability in intelligent vehicle, surveillance, security, HCI, robotics and other areas, both of these tasks have received a significant amount of research interest over the years, and a large number of different techniques has been proposed (*e.g.* see the human detection surveys by Dollár et al. (2012), Enzweiler and Gavrila (2009) and face detection surveys by Zhang and Zhang (2010), Yang et al. (2002)).

While the existing body of research on these task is very large (*e.g.* a survey by Yang et al. alone references more than 150 reported face detection approaches), a few of the most influential recent techniques are summarized below, starting with a very brief overview of statistical approach to object detection.

3.1.1.1 Generative vs. discriminative statistical object detection

Statistical approaches to object detection in images are often split into generative and discriminative (see, for example, Ng and Jordan (2001)). Generative approaches model the joint probability of object's appearance x and its class y (*i.e.* $\Pr(x,y)$). Often this is achieved by learning a class conditional probability of generating a certain object's appearance (*i.e.* $\Pr(x|y)$), and the class priors ($\Pr(y)$). Combined with the Bayes rule this approach allows to find the class $\hat{y}$ that maximizes the posterior probability $\Pr(y|x)$ for a given object's appearance x, by calculating $\hat{y} = \arg\max_y \Pr(y|x) = \arg\max_y \Pr(x,y) = \arg\max_y \Pr(x|y)\Pr(y)$. An example of an early generative approach for human detection proposed by Gavrila and Philomin (1999) is given in section 3.1.1.4.

In contrast, discriminative approaches learn the boundary that separates the object's classes, *i.e.* they directly model the (scaled) posterior distribution $\Pr(y|x)$. As argued by Vapnik (1998), generative classifiers need to solve a more complicated modelling problem to obtain $\Pr(x|y)$, even though the joint distribution serves just as an intermediate step in the posterior distribution calculation. Often such attempts run into data sparsity problems due to the "curse" of high dimensionality. For this reason, the majority of state-of-the-art human/face detectors use discriminative techniques (Dollár et al., 2012, Zhang and Zhang, 2010), in which the discriminative classifiers are combined with the human/face feature descriptors to obtain binary[1] classifiers. Both the classifiers and the features that they use are described below.

3.1.1.2 Discriminative classifiers frequently used in human detection

Widely adopted discriminative classifier learning paradigms include feed-forward multilayer neural network training using backpropagation (Werbos, 1974), adaptive weak-classifier boosting (AdaBoost, Freund and Schapire (1997)) and maximally-separating hyperplane learning using support vector machines (SVMs, Cortes and Vapnik (1995)). Since a large number of human body/face detection methods are based on these features discriminative classifiers, each of them is individually described below, starting with the support vector machines. (The human descriptor features used by these classifiers are much more diverse and are described later.)

Support Vector Machines Linear SVMs, as originally introduced by Cortes and Vapnik (1995), can be used for fast and accurate binary classification of linearly-separable data.

Given a set of training examples $A = (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_k, y_k)$ containing N-dimensional data points $\boldsymbol{x}_i$ and their classes y_i, SVMs construct an optimally separating hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$, which maximizes the distance between the hyperplane and the closest positive/negative training points (the "margin"). The boundaries of the margin are described by the support hyperplane equations $\boldsymbol{w}^T\boldsymbol{x} + b = \pm 1$ (by appropriately rescaling $\boldsymbol{w}$ and b, since it does not change the decision hyperplane), with the distance between them equal to $\frac{2}{||\boldsymbol{w}||}$ (illustrated in image a) of figure 3.1).

The parameters of the optimally separating hyperplane $\boldsymbol{w}$ and b (normal vector and offset from the origin, scaled by $||\boldsymbol{w}||$) can be found by solving the following quadratic optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}||\boldsymbol{w}||^2, \\ s.t. \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \text{ for } i = 1, ..., k, \end{aligned} \tag{3.1}$$

[1]"Human"/"non-human" or "face"/"non-face".

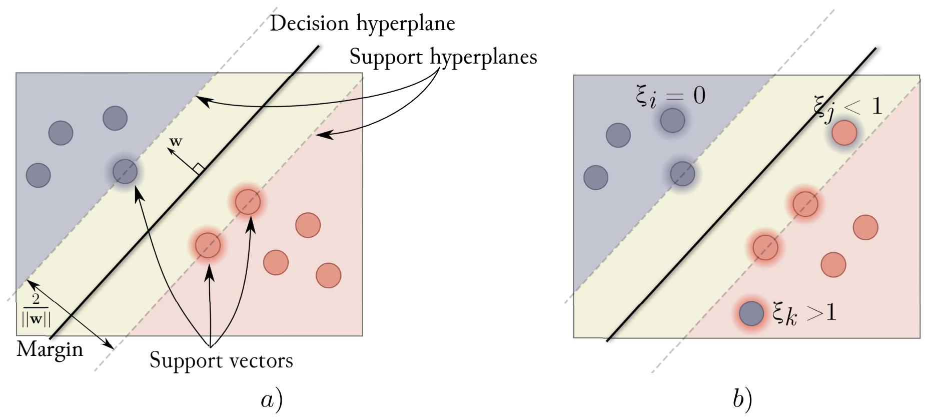


Figure 3.1: Support Vector Machines: *a)* a regular linear SVM, *b)* a soft-margin linear SVM. The decision (maximally separating) hyperplane is described by equation $\boldsymbol{w}^T\boldsymbol{x}+b=0$, support hyperplanes are described using equations $\boldsymbol{w}^T\boldsymbol{x}+b=\pm 1$.

or its dual Lagrangian form:

$$\begin{aligned} \max_{\alpha_1,\dots,\alpha_k} \quad & \sum_{i=1}^{k}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{k}\alpha_i\alpha_j(\boldsymbol{x}_i^T\boldsymbol{x}_j) \\ s.t. \quad & \alpha_i \geq 0, \text{ for } i=1,\dots,k, \text{ and} \\ & \sum_{i=1}^{k}\alpha_i y_i = 0. \end{aligned} \tag{3.2}$$

After constructing the maximum-margin hyperplane, the class of an unseen N-dimensional point $\boldsymbol{x}$ can be determined by examining on which side of the hyperplane the point lies, *i.e.* by calculating $f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}+b)$ or equivalently $f(\boldsymbol{x}) = \text{sign}(\sum_{i=1}^{k}\alpha_i y_i(\boldsymbol{x}_i^T\boldsymbol{x})+b)$. Since only support vectors will have non-zero Lagrangian parameters α_i, new data points can be very efficiently classified by SVMs.

SVMs can deal with non-linearity in two ways: by allowing "soft" margins (*i.e.* allowing some outliers to be misclassified) or by using a non-linear transformation to project the data points from the input space to a high- or infinite-dimensional feature space.

In the former case this is achieved by allowing a given point x_i to violate the margin by $\xi_i \geq 0$ (*i.e.* to have the distance to the separating hyperplane of $\frac{1-\xi_i}{||\boldsymbol{w}||} \leq \frac{1}{||\boldsymbol{w}||}$, see image *b)* in figure 3.1 for illustration). This results in the following primal form of the optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi_1,\dots,\xi_k} \quad & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{k}\xi_i, \\ s.t. \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b) \geq 1-\xi_i, \text{ for } i=1,\dots,k, \text{ and} \\ & \xi_i \geq 0, \text{ for } i=1,\dots,k, \end{aligned} \tag{3.3}$$

where C controls the trade-off between the margin size and the toleration of misclassified outliers.

In the latter case, the "kernel trick" is used. The key insight there is that the linearly non-separable data in the input space could be projected to the inner product space (with its associated norm) in which the data becomes linearly separable (see figure 3.2). This can be achieved by replacing all inner products $\boldsymbol{u}^T\boldsymbol{v}$ with the inner products in the projected space $K(\boldsymbol{u},\boldsymbol{v}) = \langle\phi(\boldsymbol{u}),\phi(\boldsymbol{v})\rangle$, where $\phi(\cdot)$ is the projection operator. It is important to note that the explicit representation for $\phi(\cdot)$ is not required, provided that $K(\boldsymbol{u},\boldsymbol{v})$ satisfies the Mercer's (1909) condition.

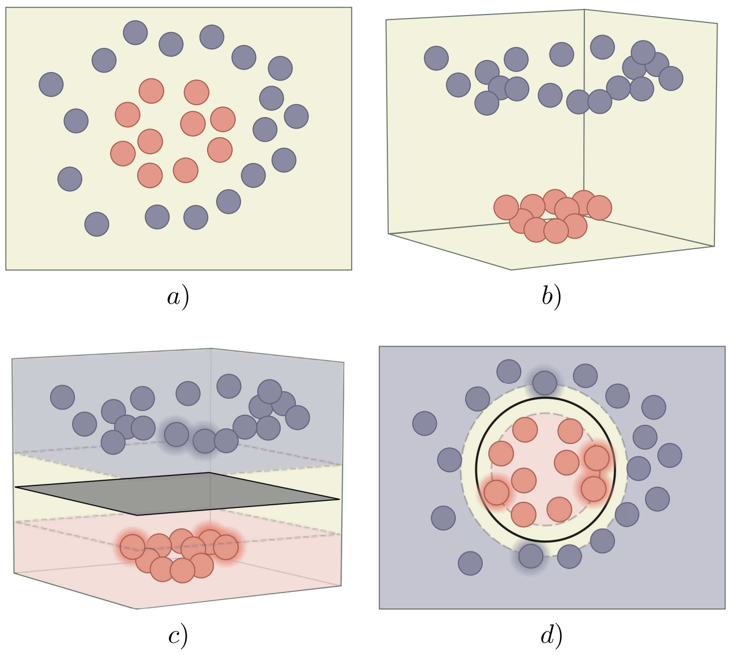


Figure 3.2: "Kernel trick" illustration using linear SVMs. The original 2-dimensional non-linearly separable input data from image $a)$ is transformed into a 3-dimensional feature space using the quadratic transformation $[x_1, x_2] \mapsto [x_1, x_2, x_1^2 + x_2^2]$, where it becomes linearly separable. The obtained decision/support hyperplanes are shown in image $c)$, and the final decision boundary (when transformed back into the input space) is shown in $d)$.

Finally, the obtained quadratic optimization problems can be solved using industry standard QP solvers (see Bottou and Lin (2007) for reference).

Weak-classifier boosting using AdaBoost A different approach of learning discriminative classifiers is proposed by Freund and Schapire (1997). In particular, Freund and Schapire proposes a method to combine a given family of "weak" classifiers (where classifiers are "weak" in a sense that they perform only marginally better than random) into a "strong" classifier through a number of training rounds, where in each round $i)$ the best weak classifier for the current training data is chosen, and $ii)$ the training example weights are decreased/increased based on their correct/incorrect classification respectively.

The final strong classifier is obtained by taking a weighted linear combination of weak classifiers, where the weights assigned to individual weak hypotheses are inversely proportional to the number of classification errors that they make.

This approach is formalized in AdaBoost algorithm (3.1.1.1) and illustrated in figure 3.3.

As proven by Schapire and Singer (1999), the training error of a strong classifier obtained using AdaBoost decreases exponentially in the number of rounds, *i.e.* using the notation from algorithm 3.1.1.1, the training error at round T is bounded by

$$\frac{1}{N}\sum_{i=1}^{N} [\![\mathrm{sign}(f(\boldsymbol{x}_i)) \neq y_i^{\pm}]\!] \leq \frac{1}{N}\sum_{i=1}^{N} \exp\left(-y_i^{\pm} f(\boldsymbol{x}_i)\right), \text{ where } y_i^{\pm} = \begin{cases} 1, & \text{if } y_i = 1, \\ -1, & \text{otherwise.} \end{cases} \tag{3.4}$$

Feed-forward multilayer neural networks A third common approach of learning discriminative classifiers involves learning the weights of artificial neural networks (ANNs). The main building block in an ANN is a single neuron, whose behaviour is defined by a weight vector $\boldsymbol{w} = [w_0, ..., w_n]^T \in \mathbb{R}^{n+1}$ and an activation function

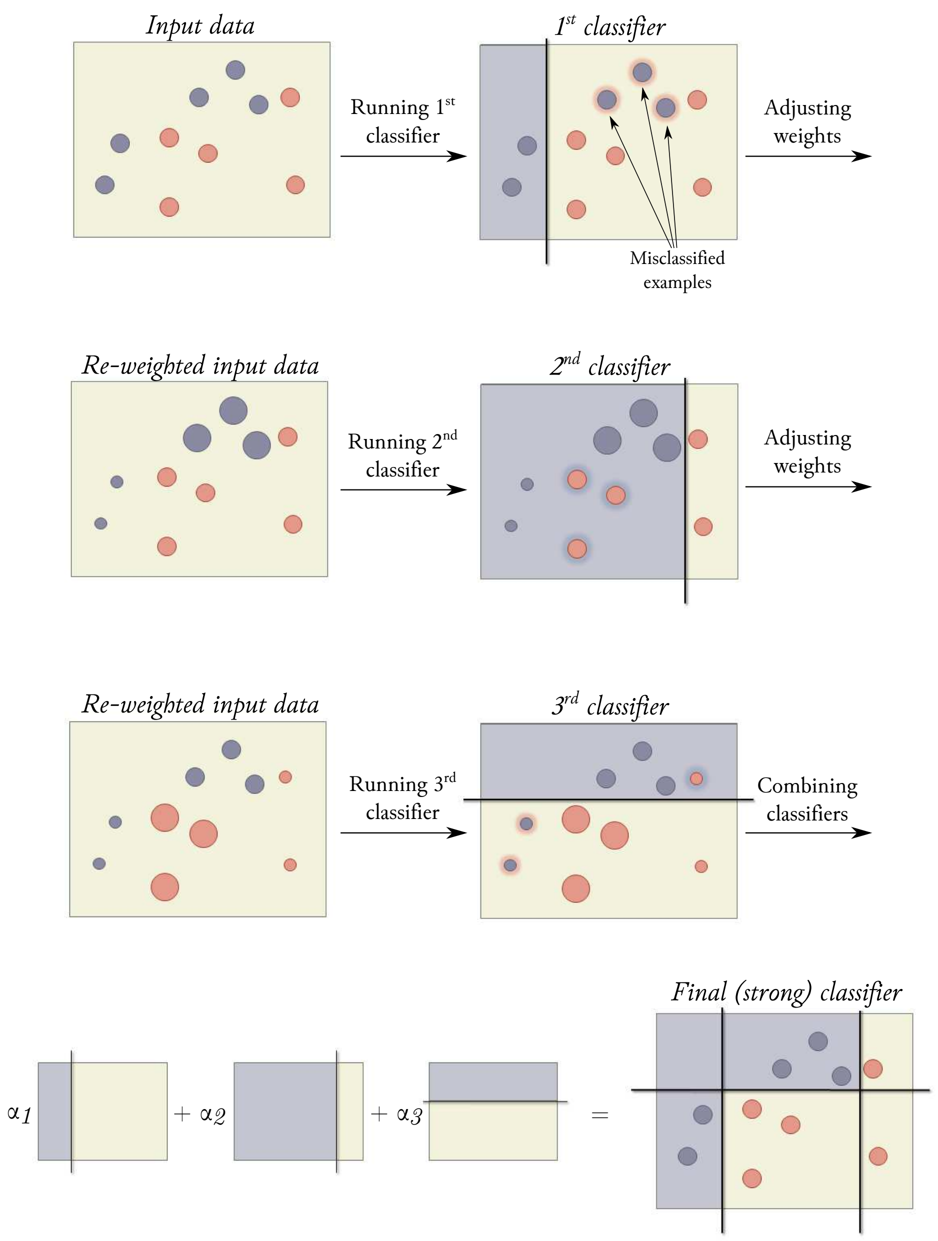


Figure 3.3: A simplified illustration of AdaBoost weak classifier boosting algorithm (3.1.1.1). In this training sequence, three weak classifiers that minimize the classification error are selected; after selecting each classifier, the remaining training examples are reweighed (increasing/decreasing the weights of incorrectly/correctly classified examples respectively). When all three classifiers are selected, a weighed linear combination of their individual thresholds is taken, yielding a final strong classifier. Adapted from Zabarauskas (2012).

Algorithm 3.1.1.1 Weak classifier boosting using *AdaBoost*. It requires N training examples given in the array $A = (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)$ (where $y_i = 0$ for a negative and $y_i = 1$ for a positive training example), a family $\mathcal{H}$ of weak classifiers and a number of rounds T to construct a strong classifier. The result of the boosting is a final strong classifier f, which can be used to classify an unseen example $\boldsymbol{x}$ based on the sign of $f(\boldsymbol{x})$.

```
AdaBoost($A, \mathcal{H}, T$)
 1  // Initialize training weights (where $m$ is the count of negative, $l$ is the count of
 2  // positive training examples).
 3  for each training example $(\boldsymbol{x}_i, y_i) \in A$
 4      if $y_i = 0$
 5          $w_{1,i} \leftarrow \frac{1}{2m}$
 6      else
 7          $w_{1,i} \leftarrow \frac{1}{2l}$

 8  for $t \leftarrow 1$ to $T$
 9      // 1. Normalize the weights:
10      for each weight $w_{t,i}$
11          $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{N} w_{t,j}}$

12      // 2. Select the best weak classifier $h_t(x) \in \mathcal{H}$ which minimizes the training error $\epsilon_t$:
13      $h_t \leftarrow \arg\min_{h \in \mathcal{H}} \sum_i w_{t,i} [\![ h(\boldsymbol{x}_i) \neq y_i ]\!]$ // where $[\![.]\!]$ is the indicator function.
14      $\epsilon_t \leftarrow \sum_i w_{t,i} [\![ h_t(\boldsymbol{x}_i) \neq y_i ]\!]$

15      // 3. Update the weights:
16      for each training example $(\boldsymbol{x}_i, y_i) \in A$
17          if $h_t(\boldsymbol{x}_i) = y_i$
18              $w_{t+1,i} \leftarrow w_{t,i} \frac{\epsilon_t}{1-\epsilon_t}$
19          else
20              $w_{t+1,i} \leftarrow w_{t,i}$

21  return $f(\boldsymbol{x}) = \sum_{t=1}^{T} (h_t(\boldsymbol{x}) - \frac{1}{2}) \log \frac{1-\epsilon_t}{\epsilon_t}$.
```

$\sigma : \mathbb{R} \to \mathbb{R}$. In particular, given an input vector $\boldsymbol{x} = [1, a_1, ..., a_n]^T$ the neuron produces an output by calculating $\sigma(\boldsymbol{x}^T \boldsymbol{w})$, as illustrated in figure 3.4[2].

A single neuron can be trained by minimizing the square error, which given a set of training examples $\{\boldsymbol{x}_i, y_i\}$ (where $\boldsymbol{x}_i \in \mathbb{R}^n$ is a vector of inputs, and $y_i \in \mathbb{R}$ is the observed output) can be defined as

$$\xi(\boldsymbol{w}) = \sum_i \left(\sigma(\boldsymbol{x}_i^T \boldsymbol{w}) - y_i \right)^2 .$$

The error can be minimized using gradient descent method by initializing the weights vector to a random vector $\boldsymbol{w}_0 \in \mathbb{R}^{n+1}$ and using the following iterative weight update rule, until the weights change by less than a predefined

[2]This subsection presents the derivations using the sigmoid activation function (*i.e.* $\sigma(x) \triangleq \frac{1}{1+\exp(-x)}$), but other activation functions, like hyperbolic tangent $\sigma(x) = \tanh(x)$, arctangent $\sigma(x) = \frac{2}{\pi} \tan^{-1}\left(\frac{\pi}{2} x\right)$, error function $\sigma(x) = \operatorname{erf}\left(\frac{\sqrt{\pi}}{2} x\right)$ or simple algebraic functions like $\sigma(x) = \frac{x}{\sqrt{1+x^2}}$ or $\sigma(x) = \frac{x}{1+|x|}$ can also be used.

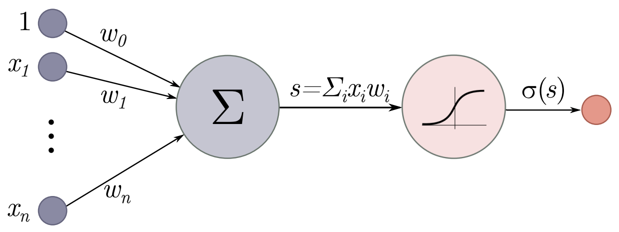


Figure 3.4: Behaviour of a neuron with the sigmoid activation function. Neuron's weights are given by $w_0, \dots, w_n$ and the inputs are given by $x_1, \dots, x_n$.

threshold:

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \eta \left. \frac{\partial \xi(\boldsymbol{w})}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}_t} \\ &= \boldsymbol{w}_t - 2\eta \sum_i \left(\sigma(\boldsymbol{x}_i^T \boldsymbol{w}_t) - y_i\right) \sigma(\boldsymbol{x}_i^T \boldsymbol{w}_t) \left(1 - \sigma(\boldsymbol{x}_i^T \boldsymbol{w}_t)\right) \boldsymbol{x}_i, \end{aligned} \tag{3.5}$$

where η is the gradient descent step size.

While a single neuron can only learn a separating hyperplane (*i.e.* it is limited to linearly separable data), a multilayer neural network can approximate more sophisticated functions. In fact, Cybenko (1989) has proved the *universal approximation theorem* for sigmoid activation functions, which states that a three-layer feed-forward neural network (with a finite number of hidden neurons) can be trained to approximate any continuous non-linear function with arbitrary precision.

The weights for multilayer feed-forward neural networks can be learned using a backpropagation algorithm (Werbos, 1974), summarized below.

Let $w_{i \to j}$ be the weight of the synapse connecting neurons i and j, let the sum of the weighed inputs of neuron j be denoted by $s_j = \sum_k z_k w_{k \to j}$ where k iterates over all neurons connected to j, and let the output of j be written as $z_j = \sigma(s_j)$, where σ is j's activation function (see figure 3.5 for illustration).

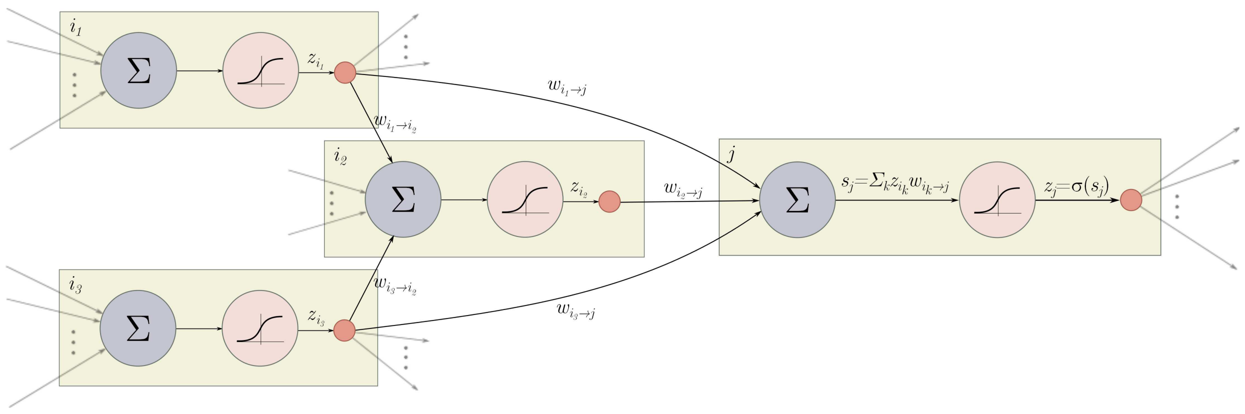


Figure 3.5: An excerpt from a multilayer feed-forward network, illustrating the notation described in the text.

Furthermore, let the output of the network be defined as $h_{\boldsymbol{w}}(\boldsymbol{x})$, where the weights vector $\boldsymbol{w}$ contains all weights in the network, *i.e.* $\boldsymbol{w} = (w_{i \to j})$ for all i, j. Then, for the same error measure

$$\xi(\boldsymbol{w}) = \sum_i \xi_i(\boldsymbol{w}) = \sum_i \left(h_{\boldsymbol{w}}(\boldsymbol{x_i}) - y_i\right)^2$$

the gradient descent update for the weights can be described using

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \sum_i \left. \frac{\partial \xi_i(\boldsymbol{w})}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}_t},$$

where

$$\frac{\partial \xi_i(\boldsymbol{w})}{\partial w_{j \to k}} = 2\left(h_{\boldsymbol{w}}(\boldsymbol{x_i}) - y_i\right) z_j \frac{\partial h_{\boldsymbol{w}}(\boldsymbol{x}_i)}{\partial s_k},$$

and

$$\frac{\partial h_{\boldsymbol{w}}(\boldsymbol{x}_i)}{\partial s_k} = \begin{cases} \sigma'(s_k), & \text{if } k \text{ is an output neuron,} \\ \sum\limits_{\substack{v \text{ receives} \\ \text{from } k}} \frac{\partial h_{\boldsymbol{w}}(\boldsymbol{x}_i)}{\partial s_v} w_{k \to v} \sigma'(s_k), & \text{otherwise.} \end{cases}$$

The error gradient can then be calculated by placing the i^{th} example at the inputs of the neural network, calculating s_k and z_k for all the nodes (the "forward-propagation" step) and working backwards from the output node calculating $\frac{\partial h_{\boldsymbol{w}}(\boldsymbol{x}_i)}{\partial s_k}$ (the "backpropagation" step).

3.1.1.3 Human/face descriptor features

The discriminative classifiers as described above use feature vectors both for training and for classification. The most common descriptor features from which these vectors are constructed are described below, while further examples of discriminative detectors based on these features are discussed in the appendix A.2.

Image intensities The simplest possible approach of obtaining the features to be used with discriminative classifiers is to use the raw image intensities.

One of the earliest advanced face detection systems developed by Rowley et al. (1998) uses this approach. In particular, Rowley et al. describe a few three-layer feed-forward neural network configurations, connected in a retinal fashion. A representative multilayer network ("network 1" in the original source) consists of:

- An input layer, which takes a 20×20 pixel size, intensity normalized, grayscale input image.
- Two copies of the hidden layer, which contain 26 neurons each. Four neurons in this layer are connected to non-overlapping 10×10 pixel regions, 16 neurons connected to non-overlapping 5×5 pixel regions and 6 neurons connected to overlapping 20×5 pixel horizontal regions.
- An output layer, which contains a single real-valued output neuron, which is connected to the outputs of all hidden layer neurons. This neuron produces a (scaled) estimate of $\Pr(\textit{face}|\textit{image})$.

Nearly 3,000 weights of "network 1" are learned in a supervised fashion: around 15,000 face images (generated by slight rotations, translations, scaling and mirroring of over a 1,000 original face images) are used as positive training examples; negative training examples (not containing faces) are bootstrapped from 120 large-resolution images of scenery, using false positive misclassifications of the neural network at each iteration of the backpropagation algorithm.

Faces are then detected using a sliding-window approach over a multiscale pyramid representation of the input image. In order to reduce the number of false positive detections, Rowley et al.'s system uses multiple neural networks in parallel (trained on different subsets of the data), combined with different schemes of arbitration. The behaviour of the overall system and the structure of "network 1" is illustrated in figure 3.6.

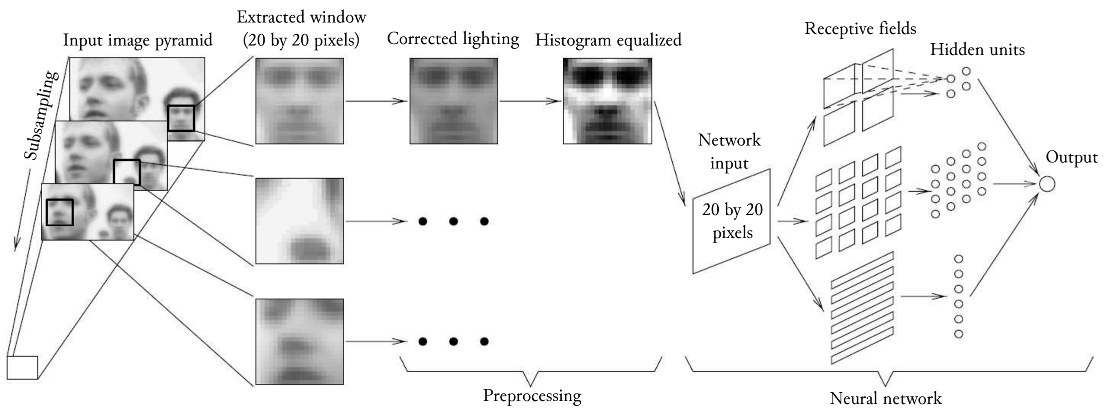


Figure 3.6: Rowley et al.'s (1998) face detection system based on feed-forward multilayer neural networks, trained using raw image intensities as the input features. Adapted from Rowley et al. (1998).

Principal components Principal Component Analysis (PCA, Pearson (1901)) is one of the best known feature extraction/dimensionality reduction techniques. Given a set of observations produced by potentially correlated variables, PCA finds the eigenvectors corresponding to the largest-magnitude eigenvalues of the data's covariance matrix (called "principal components"), and re-projects the data in these directions. In this new coordinate system, the observations are guaranteed to be uncorrelated.

Given a set of aligned human images (of equal size) PCA can be used to extract the principal components, which describe the most variance in the training data (by discarding the principal components with small eigenvalues). Then the human images can be succinctly represented in this new, reduced dimensionality space, and these representations can be used as input features for discriminative classifiers. A common way of finding these eigenvectors (principal components) and their associated eigenvalues is by using a Singular Value Decomposition (SVD).

Given a matrix $\boldsymbol{X}$ containing m data samples in the columns (with n rows representing individual features), $\boldsymbol{X}$ can be decomposed as $\boldsymbol{X} = \boldsymbol{U\Sigma V}^T$ using SVD, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices, and $\boldsymbol{\Sigma}$ is a diagonal matrix containing non-negative values.

Then $\boldsymbol{X}$'s covariance matrix $\boldsymbol{C}$ can be expressed as

$$\begin{aligned}\boldsymbol{C} &= \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^T\\ &= \frac{1}{n}(\boldsymbol{U\Sigma V}^T)(\boldsymbol{U\Sigma V}^T)^T\\ &= \frac{1}{n}\boldsymbol{U\Sigma V}^T\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\\ &= \boldsymbol{U}\left(\frac{1}{n}\boldsymbol{\Sigma}^2\right)\boldsymbol{U}^T \text{ (Using the orthonormality of } \boldsymbol{U}, \boldsymbol{V} \text{ and diagonality of } \boldsymbol{\Sigma}).\end{aligned}$$

The principal components (= eigenvectors of the covariance matrix $\boldsymbol{C}$) can be obtained by reading off non-zero columns of $\boldsymbol{U}$. The associated eigenvalues are given on the diagonal of matrix $\frac{1}{n}\boldsymbol{\Sigma}^2$.

This technique can be applied to aligned, grayscale, $M \times N$-pixel size human images by subtracting the mean image from all training images, "linearizing" individual training images by putting all pixel intensities from the image into a single $M \times N$ length column vector and combining these column vectors into the data matrix $\boldsymbol{X}$.

Turk and Pentland (1991) apply this technique in face detection by extracting the principal components (which they named "eigenfaces") from an aligned grayscale face training set. A representative example of eigenfaces is shown in figure 3.7.

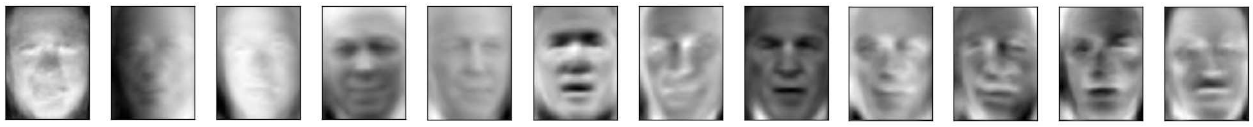


Figure 3.7: Twelve eigenfaces with the corresponding largest eigenvalues, extracted from the Labeled Faces in the Wild (Huang et al., 2007) dataset. Adapted from Pedregosa et al. (2011).

After extracting the principal components Turk and Pentland consider a reduced-dimensionality "face space", which is obtained by keeping just the top k eigenfaces with the highest corresponding eigenvalues. Human face images projected into this space stay relatively similar to the original images (since the top eigenfaces successfully capture a large amount of variance in the input image), while non-face images projected to the "face space" appear very different. This property is exploited using a concept of "face map" ξ, where each point on the face map $\xi(x, y)$ corresponds to the distance between the image window centered at (x, y) and this window's projection onto face space. The faces can then be detected by searching for the minima in the face map.

Haar wavelets Papageorgiou and Poggio (2000) propose the use of 2-dimensional Haar wavelets to encode the features of the human visual appearance.

In a single dimension Haar wavelets are defined as follows. Let j be the "resolution" of a 1-dimensional image containing 2^j pixels. Then the basic Haar wavelet at resolution j is defined as $\psi_i^j = \sqrt{2^j}\psi(2^j x - i)$ for $i = 0, ..., 2^j - 1$, where

$$\psi(x) = \begin{cases} 1, & 0 \le x < \frac{1}{2}, \\ -1, & \frac{1}{2} \le x < 1, \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, the scaling function of Haar wavelets is defined as $\phi_i^j = \sqrt{2}\phi(2^j x - i)$ for $i = 0, ..., 2^j - 1$, where

$$\phi(x) = \begin{cases} 1, & 0 \le x < 1, \\ 0, & \text{otherwise.} \end{cases}$$

Given this single-dimensional Haar wavelet definition, Papageorgiou and Poggio propose the following 2-dimensional generalizations of Haar wavelets:

- "Vertical" wavelets, obtained by taking a tensor product of a wavelet with a scaling function, $\phi_h(x, y) = \psi(x) \otimes \phi(y)$,
- "Horizontal" wavelets, obtained by taking a tensor product of a scaling function with a wavelet, $\phi_v(x, y) = \phi(x) \otimes \psi(y)$,

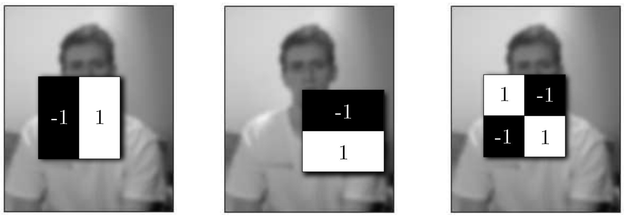


Figure 3.8: Two-dimensional Haar wavelets with square support, as described by Papageorgiou and Poggio (2000). From left to right: vertical, horizontal and diagonal 2D Haar wavelets.

- "Diagonal" wavelets, obtained by taking a tensor product of two wavelets, $\phi_d(x, y) = \psi(x) \otimes \psi(y)$.

Furthermore, they define these wavelets as always having a square support, yielding the wavelets illustrated in figure 3.8.

Finally, to obtain a richer model of humans, Papageorgiou and Poggio use an over-complete wavelet vocabulary, which is obtained by allowing a 75% percent spatial overlap between wavelets. For the 128×64 pixel window and two wavelet resolutions (32×32 and 16×16 pixels), this approach yields a vocabulary containing 1,326 wavelets.

To reduce the detector's sensitivity to lighting, Papageorgiou and Poggio calculate the Haar wavelet feature responses individually in each of the R, G, B channels and keep only the one with the highest absolute value. This yields a 1,326 feature vector representing an input image.

The detector itself is based on the SVM classifier as described above, with a quadratic kernel ($K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y} + 1)^2$). It is trained using 1,800 positive training images (containing humans) and 16,726 negative training images (guaranteed not to contain humans).

The average wavelet responses to the images in the training set (normalized and encoded in grayscale) are shown in figure 3.9.

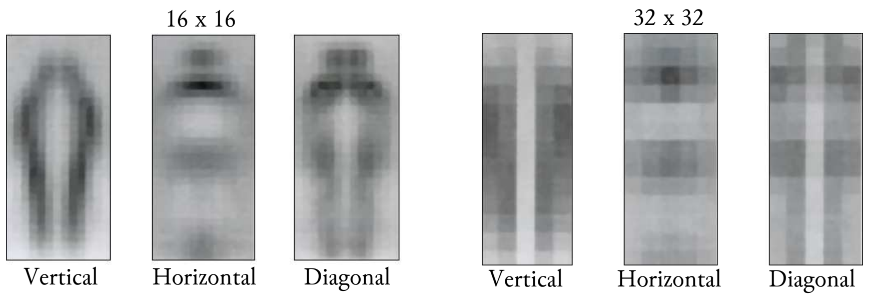


Figure 3.9: The average Haar wavelet responses (normalized and encoded in grayscale) to the images in the training set of Papageorgiou and Poggio's (2000) human detector. Reproduced from Papageorgiou and Poggio (2000).

Haar-like features Viola and Jones (2004) propose the use Haar-like features, which generalize 2-D Haar wavelets of Papageorgiou and Poggio by allowing non-square support and additional configurations (see figure 3.10).

These Haar-like features are further generalized by Lienhart and Maydt (2002), by allowing different feature orientations and adding a few more possible configurations (see figure 3.11).

The values of these features can be efficiently calculated using "integral image" representation: given a grayscale input image $I(x, y)$, the integral image is defined as $\widetilde{I}(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y')$ (see figure 3.12). Using dynamic programming the integral image can be constructed in $O(n)$ time, where n is the number of pixels in the input image.

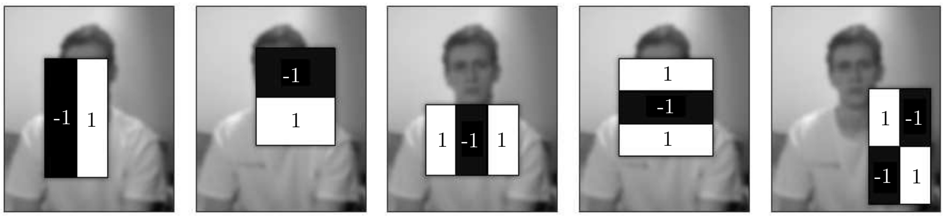


Figure 3.10: Two-rectangle, three-rectangle and four-rectangle Haar-like features, as proposed by Viola and Jones (2004).

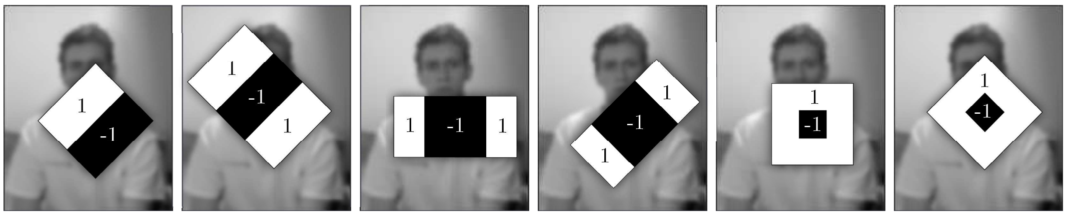


Figure 3.11: Extended set of Haar-like features, as proposed by Lienhart and Maydt (2002).

Then the integral of any rectangular area in the input image can be evaluated in four array references (see figure 3.13), meaning that *e.g.* a three-rectangle Haar-like feature response can be calculated using eight array references.

In a seminal work Viola and Jones (2004) propose a Haar-like feature based attentional face detector cascade which achieves high detection/low FP rates and performs in real-time. A given input image must be classified as a “face” by all layers in the cascade in order to have the final “face” classification; if any of the cascade layers reject the image, its processing is immediately stopped. Due to this “immediate rejection” property, each layer is required to have high detection rates, but very modest FP rates. Putting it in context, if every layer has 99% detection and 50% FP rates, then a final 15-layer cascade would have 86% detection and 0.003% FP rates.

The individual layers in this attentional cascade are obtained by combining Haar-like features with weak classification power into a strong classifier using AdaBoost (3.1.1.1). Weak classifiers are being added into the strong classifier until the layer achieves a very high detection rate (around 99%) with a modest FP rate (around 50%) on the test set. Furthermore, each of the layers in the cascade is trained on the “hard” negative training examples, which are obtained using the FP misclassifications of the earlier layers of the cascade. This means that the layers further down the cascade contain more features (and thus are more time consuming to evaluate), but at the same time, early layers can be evaluated very rapidly, especially using the integral image representation as in figure 3.13. Since faces are being detected using the sliding window approach over the input grayscale image, this allows the majority of windows (which are likely not to contain any faces) to be rejected rapidly, with the discriminative power of the cascade concentrated on face-like areas of the image.

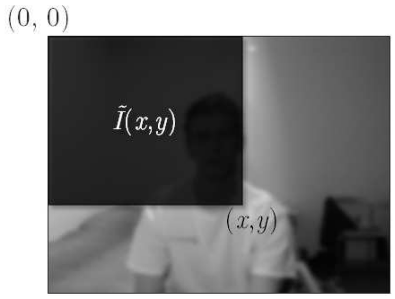


Figure 3.12: Integral image representation by Viola and Jones (2004). The value of the integral image $\widetilde{I}$ at coordinates (x, y) is defined as $\widetilde{I}(x, y) = \sum_{x' \leq x, y' \leq y} I(x, y)$, where I is the original grayscale image. Taken from Zabarauskas (2012).

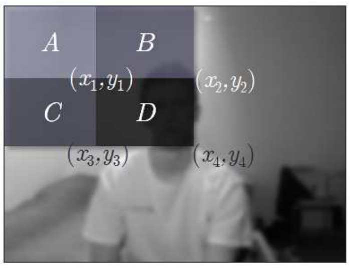


Figure 3.13: Method to rapidly calculate rectangle feature values: $D = \widetilde{I}(x_4, y_4) - \widetilde{I}(x_3, y_3) - \widetilde{I}(x_2, y_2) + \widetilde{I}(x_1, y_1)$. Taken from Zabarauskas (2012).

Local binary patterns First introduced by Ojala et al. (1996), local binary patterns (LBPs) attempt to achieve lighting invariance while retaining Haar-feature-like sensitivity to local spatial patterns. The original LBP operator defines an 8-bit value for each pixel in the input grayscale image. This value is derived from a 3×3 pixel size neighbourhood of each pixel by binarizing the neighbourhood pixel values using the center pixel value as a threshold, and arranging the results into a binary number (see figure 3.14 for illustration).

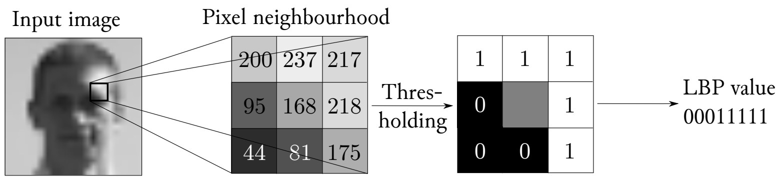


Figure 3.14: Local binary pattern (LBP) feature calculation.

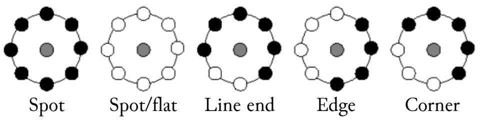


Figure 3.15: Examples of spatial pattern primitives that LBP features can detect. Adapted from Hadid et al. (2004).

As a result of this local thresholding, LBP features are to a large degree lighting-invariant; furthermore, they can detect various spatial pattern primitives (a few examples are shown in figure 3.15).

The LBP descriptors have been further extended in the work of Ojala et al. (2002), by allowing different neighbourhood sizes, and describing rotation-invariant and "uniform" families of LBPs. In the first case, the value of the operator $\text{LBP}_{P,R}(x,y)$ is obtained by thresholding P equidistant pixels on a circle of radius R centred at point (x,y) (illustrated in figure 3.16). Such operator can take one of 2^P possible values.

The "uniform" patterns are defined as the ones which have zero or two "$0 \rightarrow 1$" or "$1 \rightarrow 0$" transitions if the pattern is considered to be circular (*e.g.* patterns 00001100, 11100000 and 11111111 would be considered uniform, but a pattern 00101000 would not). This family is denoted $\text{LBP}^{\text{u2}}_{P,R}$; in the case of $P = 8$ there are 58 such patterns.

Similarly, rotation-invariant LBP patterns are defined as

$$\text{LBP}^{\text{ri}}_{P,R} = \min\{\text{ror}(\text{LBP}_{P,R}, i) | i = 0, ..., P-1\},$$

where $\text{ror}(x,y)$ is a rotate-right operator which rotates a binary input x by y bits to the right. In this case, patterns $00001111_2 = 15_{10}, 00011110_2 = 30_{10}$ and $11100001_2 = 225_{10}$ would be mapped to the same rotation-invariant pattern $00001111_2 = 15_{10}$.

These features are used in the face detector by Hadid et al. (2004). They use a standard sliding window approach, where 19×19 pixel size window is scanned over the image pyramid (with each layer subsampled at the scale 1.2). To obtain a face representation using LBPs, Hadid et al. divide the 19×19 window into 9 overlapping regions of 10×10 pixels. In each region a $\text{LBP}_{4,1}$ operator is used to obtain a 16-bin histogram of LBP values. These histograms are concatenated into a single 144-bin histogram. Furthermore, a $\text{LBP}^{\text{u2}}_{8,1}$ operator is applied to the whole 19×19 pixel image, and a 59-bin histogram is assembled (putting all non-uniform patterns into a single bin). The combined 203-bin histogram is used as a feature vector representing the face.

For the actual classifier that determines the presence/absence of a face in a given window, Hadid et al. use an SVM classifier with the quadratic kernel, trained using 6,000 face images and 14,560 bootstrapped non-face patterns.

Another LBP-based face detector is described by Zhang et al. (2007), where they extend the basic $\text{LBP}_{P,R}$ features

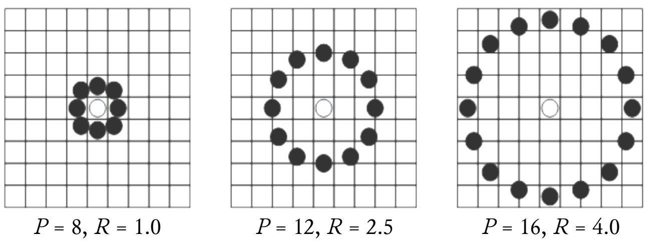


Figure 3.16: Examples of extended LBP feature set. Adapted from Ojala et al. (2002).

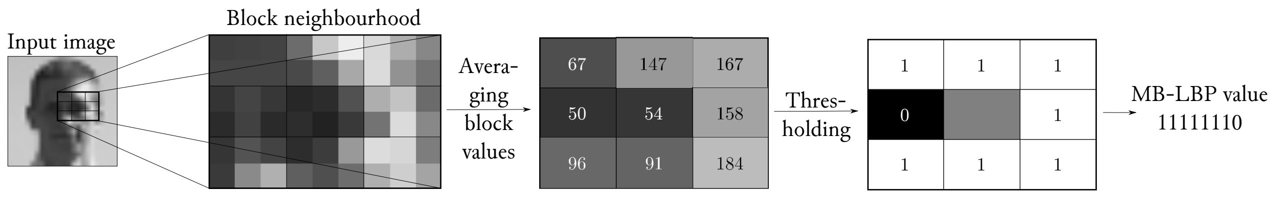


Figure 3.17: Multiple block local binary pattern (MB-LBP) feature calculation.

to deal with multiple pixel blocks, instead of individual pixel neighbourhoods (illustrated in figure 3.17). These multiple block LBP (MB-LBP) features are used in decision tree weak classifiers, which are boosted into a strong classifier using AdaBoost.

Local receptive fields Extraction of the most important non-adaptive features can be viewed as an optimization problem *w.r.t.* classification task. Analogously, finding the feature sets that adapt to the underlying training data set can be seen as part of this optimization problem (Enzweiler and Gavrila, 2009).

Adaptive local receptive fields (LRFs) introduced by Fukushima (1980) provide a way to learn the relevant features during the training using a three-layer feed-forward neural network with the following layer contents (graphically illustrated in figure 3.18):

- The first (input) layer has one neuron for each pixel of the input grayscale image I.
- The second (hidden) layer is composed of N "branches" $B_{i=1,\dots,N}$, where each neuron in every branch is connected to a fixed-size local region in the input layer, called the neuron's local receptive field. Wöhler and Anlauf (1999) who first described LRF/ANNs for human tracking, used 9×9 pixel size LRFs in the spatial dimension.

  The LRF locations of individual neurons in the same branch are non-exclusive, *i.e.* they are allowed to overlap. Furthermore, each neuron in a given branch B_i shares the same set of weights W_i, where $|W_i| =$ *number of pixels in the LRF*. Effectively, these two conditions ensure that each branch encodes some translation-invariant local feature of the image.
- The third and final (output) layer consists of two fully-connected neurons, representing the scaled posterior probability estimates for human/non-human classes respectively.

This neural network can be trained using a standard backpropagation-like approach, using the gradient descent optimization (see Wöhler and Anlauf (1999) for more details). After training, the weights of each branch in the

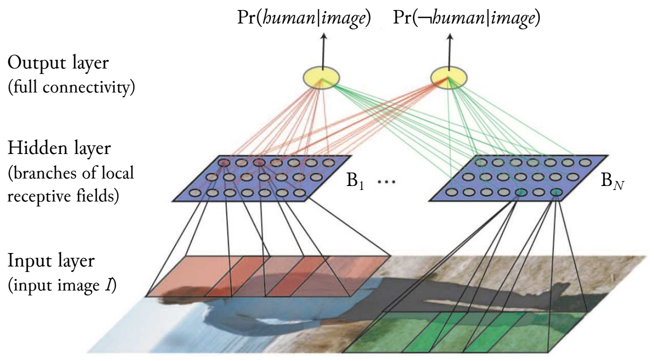

Pr(human|image)
Pr(¬human|image)
Output layer (full connectivity)
Hidden layer (branches of local receptive fields)
Input layer (input image I)
B1 ...
BN

Figure 3.18: A three-layer feed-forward neural network structure for local receptive field learning. Adapted from Enzweiler and Gavrila (2009).

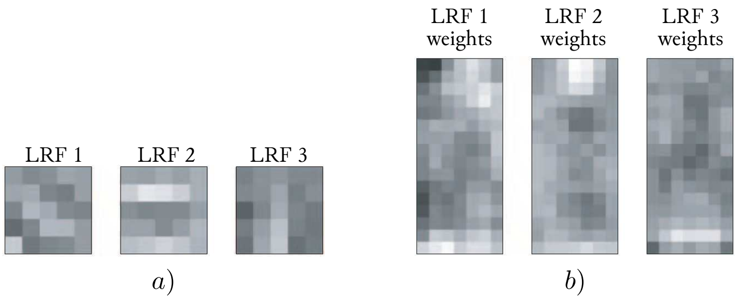


Figure 3.19: Three example 5×5 pixel LRF features with the corresponding regions of their discriminative power. Image $a)$ contains the hidden layer weights for each of the three LRF features (brighter rectangles correspond to higher weights). Image $b)$ contains the weights from the "human" detection neuron in the output layer corresponding to each of the LRF features on the left, indicating the regions where these LRFs have most discriminative power for the "human" class. Adapted from Enzweiler and Gavrila (2009).

hidden layer can be extracted to be used as features in generic classification methods. Three examples of extracted LRFs are visualized in figure 3.19.

Interestingly, Munder and Gavrila (2006) report that using these extracted features in a SVM classifier with a quadratic kernel achieves better human detection rates than using them in ANNs.

Histogram of oriented gradients (HOGs) The human detector features presented by Dalal and Triggs (2005) are based on the idea that the localized distributions of edge directions have enough information to represent the shape of an object, while being relatively invariant to local geometric transformations.

These distributions are constructed using the following procedure:

1. A gradient of a given grayscale input image I is calculated.

   To achieve this, the original image I is convolved with $G_h = [-1\ 0\ 1]$ and $G_v = [-1\ 0\ 1]^T$ filter kernels to obtain discrete approximations of horizontal and vertical derivatives of the image ($I_{\partial x} = I * G_h$ and $I_{\partial y} = I * G_v$ respectively). Then, the image gradient's magnitude and orientation images are calculated using $|\nabla(x,y)| = \sqrt{I_{\partial x}(x,y)^2 + I_{\partial y}(x,y)^2}$ and $\nabla_o(x,y) = \tan^{-1} \frac{I_{\partial y}(x,y)}{I_{\partial x}(x,y)}$ (this process is illustrated in figure 3.20).

2. The detector window is divided into rectangular (R-HOG) or circular (C-HOG) cells (Dalal and Triggs suggest the use of 8×8 pixel sized rectangular cells for the human detection task).

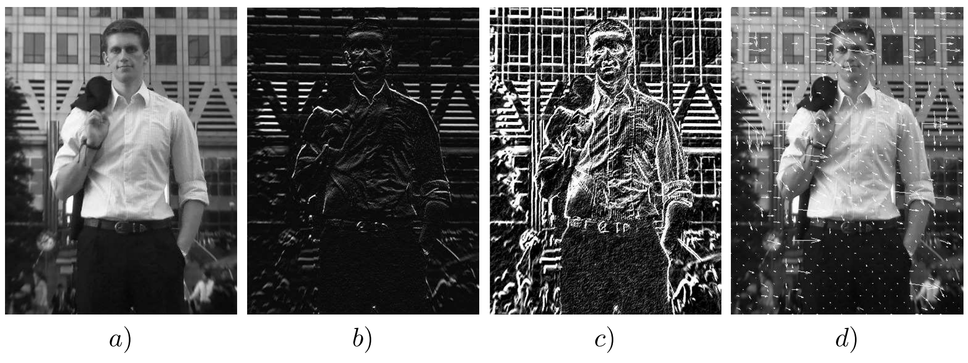


Figure 3.20: Image gradient calculation. Image $a)$ shows the original input I, image $b)$ shows the horizontal derivative $I_{\partial x}$, image $c)$ shows the gradient's magnitude $|\nabla(x,y)|$, and image $d)$ shows the combined gradient $\nabla(x,y)$.

3. Within each cell a discretized histogram of edge orientations is calculated (using edge's magnitude as its contribution to the histogram). Dalal and Triggs recommend the use of 9 orientation bins spaced at 20° over 0°–180° (ignoring the gradient's sign).

4. To provide better invariance to changes in the illumination, cell orientation histograms are normalized using overlapping blocks, which in the recommended implementation occupy 16×16 pixels and contain 2×2 cells. Due to the block overlap, each cell contributes multiple components to the final feature vector (each normalized in a different block).

The whole procedure of calculating histogram of oriented gradients features is graphically summarized in figure 3.21.

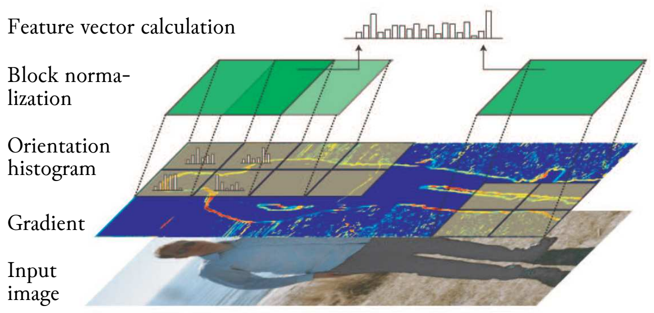


Figure 3.21: Histogram of oriented gradients (HOG) feature calculation. Adapted from Enzweiler and Gavrila (2009).

To construct a sliding-window human detector based on HOG features Dalal and Triggs train a two-class linear SVM using 1,239 positive training examples (together with their vertical mirror images) and 12,180 negative training images (guaranteed not to contain people). In the spirit of Viola and Jones (2001) detector cascade, Dalal and Triggs retrain the SVM using misclassifications from the first training round ("hard" training examples, together with the original negative training images) to improve the final classifier's performance.

HOG features are also quite similar to the scale-invariant feature transform descriptors (SIFT, Lowe (1999)). The key differences between these two types of descriptors are that i) SIFT descriptors are computed only at the image key points (as opposed to all cells in a grid), ii) are pre-processed after calculation to achieve some degree of scale/rotation invariance, and iii) are most often used for object recognition, not classification.

Shapelets In an approach that combines LRF and HOG descriptor ideas, Sabzmeydani and Mori (2007) present a way to automatically learn gradient-based human descriptor features.

Their method consists of three steps:

1. The "low-level" features are obtained by calculating absolute gradient responses of the input image in four different directions, and averaging these responses in the neighbourhood of each pixel. Formally this is achieved by convolving the input image I with the directional gradient kernel G_d[3], taking the absolute values of gradient magnitudes and convolving the result with the box filter B (a 5×5 matrix containing $\frac{1}{25}$ in all entries), *i.e.*

$$S_d = |I * G_d| * B,$$

where S_d is the resulting low-level feature.

[3]Horizontal G_h and vertical G_v kernel examples were presented in the description of HOG features; Sabzmeydani and Mori also use two diagonal kernels.

2. The detector window is discretized into smaller sub-windows (Sabzmeydani and Mori propose the sub-window sizes of 5×5, 10×10 and 15×15 pixels), and AdaBoost algorithm (3.1.1.1) is used to combine all low-level features $S_d(x, y)$ from a given sub-window i into a strong-classifier $f_i(x, y)$, called a "shapelet". These shapelet features are obtained for each sub-window i.

   The most important low-level features selected in all shapelets are shown in figure 3.22.

3. Once more, AdaBoost algorithm is used to combine the obtained shapelets $f_i(x, y)$ into a final human detector. The low-level features present in the shapelets selected by the final classifier are illustrated in figure 3.23.

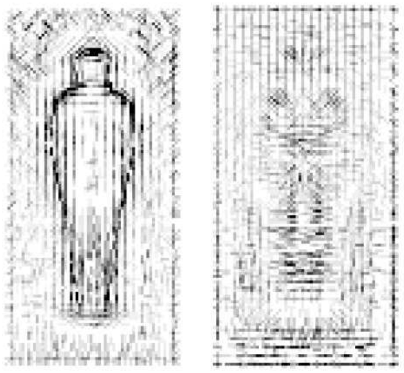


Figure 3.22: The most discriminative low-level features selected using AdaBoost in individual shapelet training. The image on the left shows the positive parity low-level features weighed by their contributions to all shapelets, the image on the right shows the negative parity low-level features. Taken from Sabzmeydani and Mori (2007).

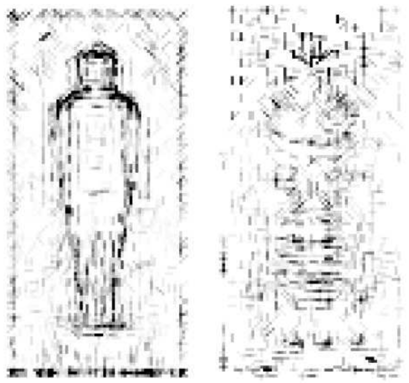


Figure 3.23: The low-level features present in the shapelets selected using AdaBoost in the final human detector. The image on the left shows the features present in the positive parity shapelets, the image on the right shows the features in the negative parity shapelets. Taken from Sabzmeydani and Mori (2007).

3.1.1.4 Other approaches for human/face detection in images

Human detection using a hierarchy of shape templates In Gavrila and Philomin's (1999) approach a discrete hierarchy of human shape templates is constructed automatically (based on the shape similarity) in a bottom-up fashion, from a library of exemplar human shapes (using around 1,000 of manually assembled shape contours, replicated at 5 scales).

At each layer of the hierarchy, the simulated annealing algorithm (Kirkpatrick et al., 1983) is used to cluster a set of exemplar shapes $t_1, ..., t_N$ into K partitions $S_1, ..., S_K$, *s.t.* the objective function

$$\sum_{k=1}^{K} \max_{t_i \in S_k} D_{\mathit{chamfer}}(t_i, p_k)$$

is minimized. In this objective function, p_k is the "prototype" exemplar shape, which has the smallest maximum distance to other shapes in the cluster, and D_{chamfer} is the chamfer distance[4]. These prototype shapes are combined into the next layer of the hierarchy, and so on. A partial view of the constructed three-level hierarchy is shown in the left side of figure 3.24.

To use this off-line shape hierarchy in an on-line matching, Gavrila and Philomin start by extracting and thresholding the edges in a given input image (using 8 discrete edge orientations) to obtain a binary "feature" image.

[4]The average distance between each feature point on one shape and the nearest feature point on the other shape (Borgefors, 1986).

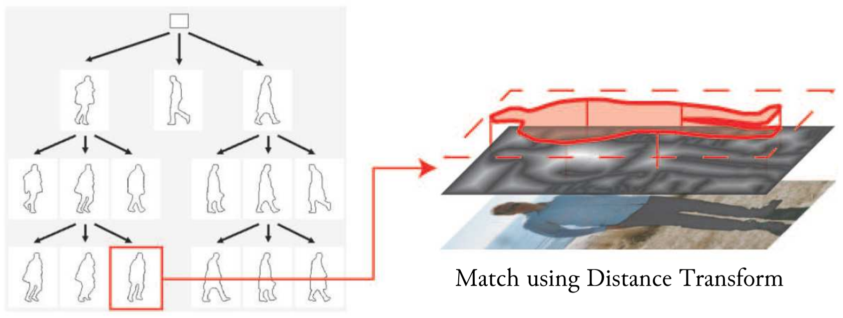


Figure 3.24: A partial human shape hierarchy, obtained by off-line clustering of human shape exemplars based on their similarity, and the subsequent on-line shape matching using distance transform. Adapted from Enzweiler and Gavrila (2009).

Then the image's distance transform (DT) is calculated (see figure 3.25 for illustration) and the image is subdivided into a coarse grid.

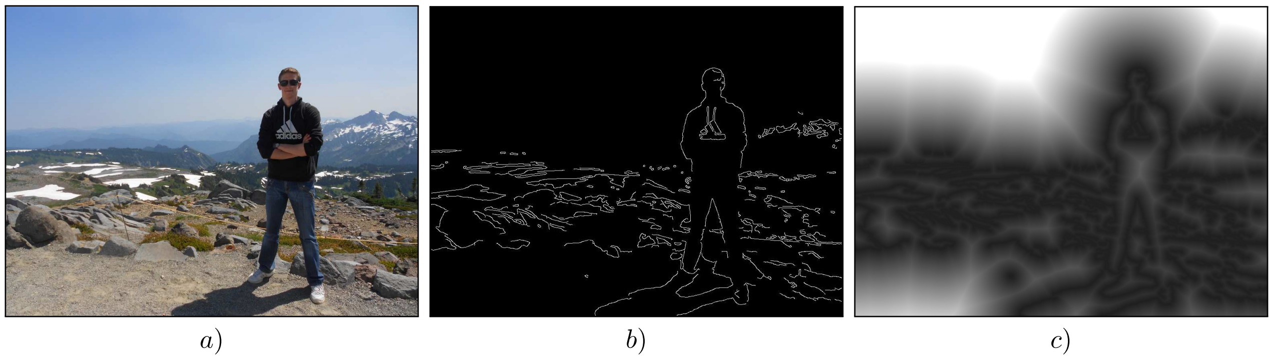


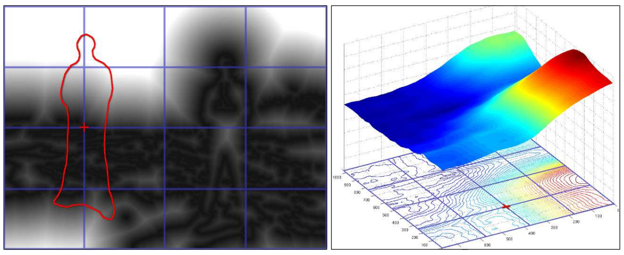
Figure 3.25: Distance transform: the original input image is shown in part a), the detected edges using a Canny (1986) edge detector are shown in part b), and the resulting distance transform is shown in part c).

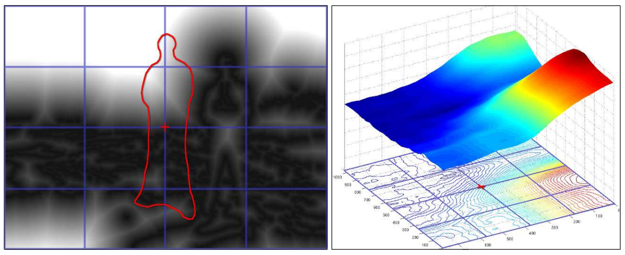
The chamfer distance is calculated between the root shape of the hierarchy and the DT image at each point on the coarse grid. If this distance is smaller than a pre-set threshold then the point's neighbourhood is searched on a finer-scale scale grid using chamfer distances to each of the child shapes. If the chamfer distance for any of the child shapes is under a distance threshold for any of the points on the finer grid, the grid is further subdivided and the shapes from the next level of the hierarchy are searched. This process is continued in a depth-first search manner until leaf-level shapes from the hierarchy are matched or all positions on the coarse grid are exhausted, as illustrated in figure 3.26.

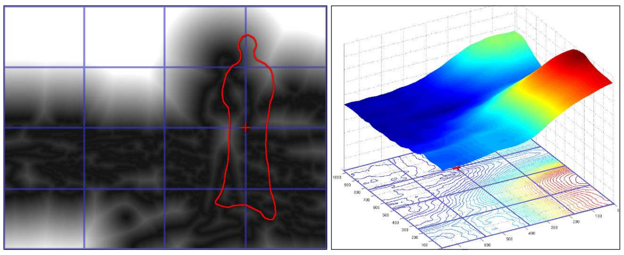
This discrete human shape modelling approach has been later extended to a fully-probabilistic Bayesian framework by Gavrila (2007). Other notable generative extensions include continuous shape modelling and inclusion of texture information (see *e.g.* Munder et al. (2008)).

Parts-based detection Besides focusing on new types of classifiers/human descriptor features, some researchers attempted to break down the problem of human detection into the problems of i) detecting individual body parts, and ii) combining these detections into a single "human"/"non-human" prediction. Two main types of decompositions have been proposed: codebook representations, which represent humans as local codebook feature assemblies, and semantically-motivated decompositions into anatomical body parts like head, arms, legs and so on.

An example of the former (codebook) approach is described by Leibe et al. (2005), in which the codebook is built by applying a Difference-of-Gaussians (DoG) operator to extract the image patches with the size of at least 3σ,


Step 1: coarse matching


Step 2: coarse matching


Step 3: coarse matching

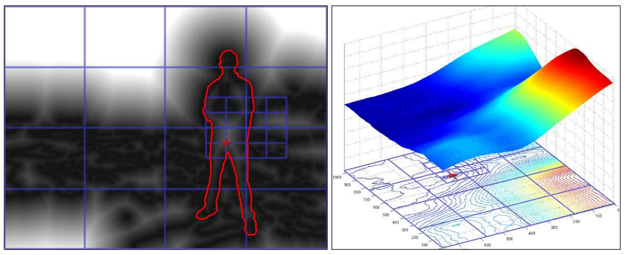

Step 4: fine matching

Figure 3.26: Hierarchical shape matching using chamfer distance calculation with the distance transform image. Each of the images show both the obtained distance transform (DT) image from figure 3.25 and the normalized chamfer distance between the DT image and the shape being matched. The first three steps illustrate the similarity search on a coarse grid/rough human shape, which switches to a search on a finer-scale grid/shape when the similarity threshold is reached, as illustrated in the step 4.

which are then clustered using an agglomerative scheme. The cluster centroids are used as codebook entries as local object structure descriptors.

Mikolajczyk et al. (2004) describe an example of the latter approach, in which they use AdaBoost to train frontal and profile head/face, frontal/profile upper body and leg detectors, using simple Laplacian/gradient-orientation based features. Then, a joint likelihood of body part configurations is modelled using the knowledge about the geometric relations between body parts.

Similar approaches have also been proposed in face detection. For example, Heisele et al. (2001) describe a parts-based face detector which uses a two-level hierarchy of SVMs: the first level of this hierarchy contains 14 linear SVM classifiers trained to detect individual face components, while the second level linear SVM acts as a geometrical classifier.

Human skin colour modelling A number of early face detection approaches attempted to detect the faces by considering human skin-like coloured patches of the input image. An example of such system is described by Yang and Ahuja (1998). In this system the skin distribution model is approximated by bivariate Gaussian distribution, fitted to a histogram of around 500 human skin images in CIE LUV colour space, with the luminance coordinate discarded. The pixel is determined to be generated by the skin if its probability is greater than $\frac{1}{2}$ under the fitted Gaussian distribution. To detect the faces, the input image is segmented into similar colour patches and the patches with less than 70% human skin colour pixels are rejected. The remaining patches are merged together into elliptical shapes, which are further verified using geometric constraints (ratio of major to minor axes) and presence of darker regions/holes in the ellipse. All shapes passing the verification are classified as a faces.

However, more recent face detection systems use the pixel's colour similarity to skin colour mostly to complement more sophisticated human/face descriptors. For example, a neural-network based face detection system by Feraund et al. (2001) uses explicit color space thresholding in YUV color space to define the skin color region. Pixels outside this region are ignored, reducing the input image area which needs to be searched for faces using the neural network.

A large number of different approaches have been proposed to model the skin colour distribution (*e.g.* see survey by Kakumanu et al. (2007)). These approaches include explicit thresholding of the color space, modelling the skin histograms using naïve Bayes classifiers, fitting a single or a mixture of Gaussians, using feed-forward multilayer neural networks, maximum entropy models, Bayesian networks and so on.

In the skin detection work by Jones and Rehg (2002) (further discussed in 3.3.1.3), a particularly large dataset was assembled for supervised skin model training. Their dataset contains 4,675 skin and 8,965 non-skin images, with over 80 million skin and 860 million non-skin pixels in total.

Jones and Rehg compared the performance of histogram and Gaussian mixture models (GMMs) when trained on such large-scale datasets. Interestingly, they discovered that histogram models outperform GMMs both in skin detection accuracy and in computational cost, achieving state-of-the-art performance.

A sample image classification into skin/non-skin areas using a naïve Bayesian classifier trained using Jones and Rehg dataset is illustrated in figure 3.27.

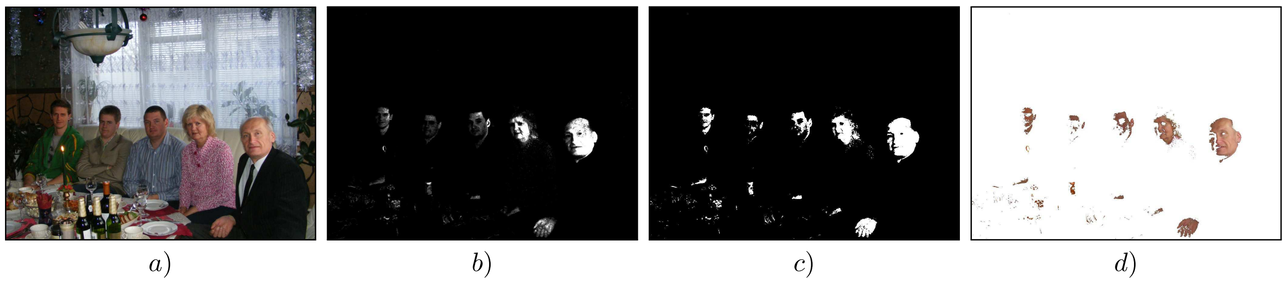


Figure 3.27: Output from naïve Bayesian skin classifier trained using Jones and Rehg's (2002) dataset. Image $a)$ shows the original input, image $b)$ shows $\frac{\Pr(skin|rgb)}{\Pr(\neg skin|rgb)}$, this ratio is thresholded in image $c)$ obtaining a binary mask, and the original image filtered using this binary mask is shown in $d)$.

3.1.2 Survey of depth-based human body/head detection methods

Early depth-based human detection work was based on range inputs from stereo cameras, time-of-flight cameras or scanning laser range finders. With the recent advent of affordable, structured-light based RGB-D cameras like Microsoft Kinect (introduced in November 2010), ASUS Xtion Pro Live (introduced in September 2011) or PrimeSense Carmine (introduced in December 2012), a number of RGB-D data based human/head detection methods have been proposed. The most influential methods are summarized below, starting with the RGB-D based human/head descriptor features used in discriminative classification frameworks.

3.1.2.1 Human/head descriptor features

Histogram of oriented depths/histogram of depth differences Spinello and Arras (2011) and Wu et al. (2011)[5] present a three-dimensional extension to histogram of oriented gradients, named histogram of oriented depths (HOD) and histogram of depth differences (HDD) respectively.

HOG/HDD features are computed from the metric depth image $D_m(x, y)$ which is obtained from the raw depth image $D(x, y)$ by calculating $D_m(x, y) = \frac{1}{a+b\times D(x,y)}$, where a, b are the intrinsic parameters of the RGB-D

[5]Both authors derived HOD/HDD features independently around the same time.

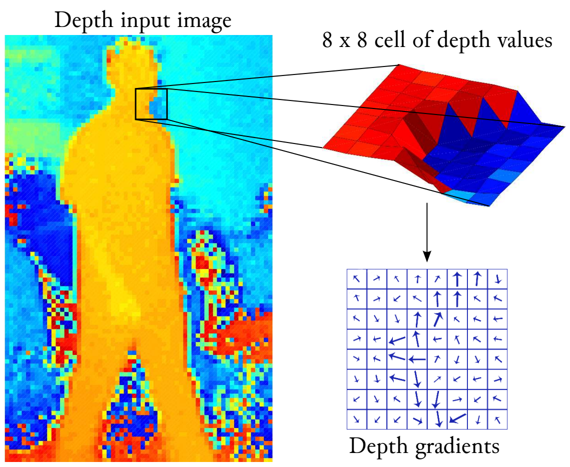


Figure 3.28: Histogram of oriented depths (HOD)/histogram of depth differences (HDD) calculation. Adapted from Wu et al. (2011).

sensor estimated during the factory or later calibration. After this depth image conversion, the HOD/HDD values are calculated from $D_m(x, y)$ in a similar way to HOGs:

1. Detector's window is subdivided into individual rectangular cells (similar to R-HOG),
2. Oriented depth gradients are calculated in each cell (as illustrated in figure 3.28) and collected into single-dimensional histograms,
3. Histograms are normalized within fixed-size blocks containing multiple cells.

The simplest classifier that uses HDD features is described by Wu et al. They train a SVM with a linear kernel on 4,637 depth images of humans, and 14,199 negative training images. For HDD feature calculation, they use 8×8 pixel size cells, and overlapping 2×2 cell blocks (with 8 pixel stride). Histogram is calculated in 9-bins, spaced equally at $40°$ over the interval $[0°, 360°)$, and histograms within individual blocks are normalized using L_2-Hys normalization measure. Then the histograms in individual cells are concatenated yielding a 3,780-dimensional SVM feature vector for a 64×128 pixel size sliding window detector.

In the approach proposed by Spinello and Arras, two linear SVMs are trained for HOG and HOD features individually, using 1,030 RGB-D samples containing people and 5,000 negative training samples (randomly selected from a background RGB-D dataset, guaranteed not to contain people). The posterior probability of human class is approximated by fitting a sigmoid function to the decision functions of both SVMs using the method of Platt (2000).

Given the probabilities $\Pr(\mathit{human}|I_D)$, $\Pr(\mathit{human}|I_G)$ (obtained from HOD/HOG detectors respectively) Spinello and Arras define the combined probability of human detection as

$$\Pr(\mathit{human}|I) = k \Pr(\mathit{human}|I_G) + (1 - k) \Pr(\mathit{human}|I_D),$$

where k is defined as the ratio of false negatives between the HOD and HOG detectors at the equal error rate point in the validation set.

Local ternary and simplified local ternary patterns In an attempt to improve the local binary pattern discrimination ability and to reduce their sensitivity to noise in uniform illumination areas of the image, Tan and Triggs (2010) propose local ternary patterns (LTPs). Given a grayscale image I, the value of the LTP feature at image coordinates (x, y) is derived by concatenating ternary responses

$$t_{(x',y')}(x,y) = \begin{cases} 1, & I(x',y') \geq I(x,y) + \delta, \\ 0, & |I(x',y') - I(x,y)| < \delta, \\ -1, & I(x',y') \leq I(x,y) - \delta, \end{cases}$$

where (x', y') ranges over (x, y) neighbours $\{(x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)\}$ and δ is the user specified threshold. An example of a LTP calculation is given in figure 3.29.

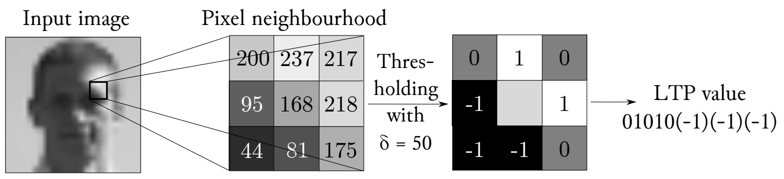


Figure 3.29: Local ternary pattern (LTP) feature calculation.

Yu et al. (2012) further extend LTP features for the use of human detection in depth images. Their proposed simplified local ternary patterns (SLTPs) combine LTPs and local first-order discrete derivatives in 8 spatial directions equally spaced between $[0, 2\pi)$. In particular, given the depth image D, the value of the SLTP centered at coordinates (x, y) is defined as

$$\text{SLTP}(x, y) = (t_x, t_y),$$

where

$$t_x = \begin{cases} 1, & \Delta_x \geq \delta, \\ 0, & |\Delta_x| < \delta, \\ -1, & \Delta_x \leq -\delta, \end{cases} = \begin{cases} 1, & D(x+1,y) \geq D(x-1,y) + \delta, \\ 0, & |D(x+1,y) - D(x-1,y)| < \delta, \\ -1, & D(x+1,y) \leq D(x-1,y) - \delta, \end{cases}$$

and

$$t_y = \begin{cases} 1, & \Delta_y \geq \delta, \\ 0, & |\Delta_y| < \delta, \\ -1, & \Delta_y \leq -\delta, \end{cases} = \begin{cases} 1, & D(x,y+1) \geq D(x,y-1) + \delta, \\ 0, & |D(x,y+1) - D(x,y-1)| < \delta, \\ -1, & D(x,y+1) \leq D(x,y-1) - \delta. \end{cases}$$

As in LTPs, δ is a user specified threshold.

An illustration of SLTP features extracted from a depth image is shown in figure 3.30.

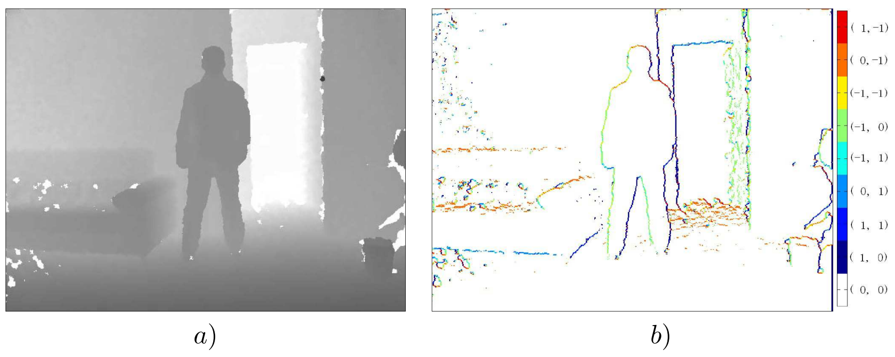


Figure 3.30: Simplified local ternary pattern (SLTP) feature calculation in depth images. Image a) shows the original depth image (where depth is indicated by the pixel intensity); the SLTPs extracted from a) are shown in image b), where different SLTPs are indicated by different colours. Note that the set of all possible SLTPs is much smaller than the set of all possible LTPs ($3^2 = 9$ *v.s.* $3^8 = 6561$ respectively). Taken from Yu et al. (2012).

For the actual human detection, Yu et al. (2012) use the sliding-window detector (with the scale factor of 1.1), implemented using linear SVMs. To obtain the feature vector that describes the window, Yu et al. subdivide the 64×128 pixel size window into 8×8 pixel non-overlapping blocks and assemble 9-bin single-dimensional histograms of SLTP features in these blocks. The histograms are then concatenated, yielding the final feature vector (this process is illustrated in figure 3.31). The SVM is trained using 4,268 manually tagged training examples containing humans in diverse postures and 59,508 negative training examples.

Since SLTP features are derived from regular LTP features, Yu et al. (2012) also evaluate an analogue human detector based on the LTPs (using 118 bins for LTP histograms). Based on their evaluation, the SLTP-based approach yields better results (lower miss rates for every "false positive per window" data point).

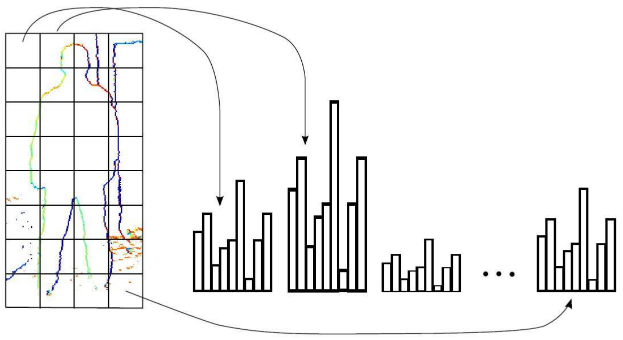


Figure 3.31: SLTP feature vector calculation in Yu et al.'s (2012) approach. First the detector's window is divided into non-overlapping blocks, then the histograms of SLTP features in those blocks are calculated. Finally, the obtained histograms are concatenated yielding the combined feature vector. Taken from Yu et al. (2012).

Local surface normals Hegger et al. (2012) propose another depth-based human descriptor, called the local surface normals (LSN). As indicated by the name, this descriptor is calculated by fitting a plane to the k-nearest neighbours of each point in the point cloud, and taking the normal vector of this plane (illustrated in figure 3.32). These normal vectors are assembled into a LSN histogram, which serves as a final feature vector.

Holz et al. (2011) present a computationally-efficient way to calculate LSNs, based on the insight illustrated in figure 3.32, *viz.* the fact that a surface normal vector to a plane can be calculated by taking the cross product of two vectors on that plane (in Holz et al.'s approach called the tangential vectors). Their method works as follows:

1. Given the input depth image $D(x, y)$, the horizontal and vertical tangential vector maps are created by calculating

$$T_h(x, y) = [x + 1, y, D(x + 1, y)]^T - [x - 1, y, D(x - 1, y)]^T,$$
$$T_v(x, y) = [x, y + 1, D(x, y + 1)]^T - [x, y - 1, D(x, y - 1)]^T.$$

2. For each of the Cartesian coordinates and each of the maps T_h, T_v an integral image (as illustrated in figure 3.12) is calculated:

$$I_{d,c}(x, y) = \sum_{\substack{x' \le x \\ y' \le y}} \left[T_d(x', y')\right]_c,$$

where $d \in \{h, v\}$ and $c \in \{x, y, z\}$.

3. For every pixel (x, y) in the depth image and any neighbourhood size k, the average tangential vectors in horizontal and vertical directions ($\boldsymbol{t}_h(x, y)$ and $\boldsymbol{t}_v(x, y)$ respectively) can be calculated using the integral

images of tangential vector maps as illustrated in figure 3.13. More specifically,

$$[\boldsymbol{t}_d(x,y)]_c = \frac{1}{k^2}(I_{d,c}(x+0.5k, y+0.5k) - I_{d,c}(x-0.5k, y+0.5k) - I_{d,c}(x+0.5k, y-0.5k) + I_{d,c}(x-0.5k, y-0.5k)),$$

for $d \in \{h, v\}$ and $c \in \{x, y, z\}$. Note, that this calculation takes constant time for each of the depth points, since it requires $2 \times 4 \times 3$ memory accesses.

4. Finally, the local surface normal vector is given by $\boldsymbol{n}(x,y) = \boldsymbol{t}_h(x,y) \times \boldsymbol{t}_v(x,y)$.

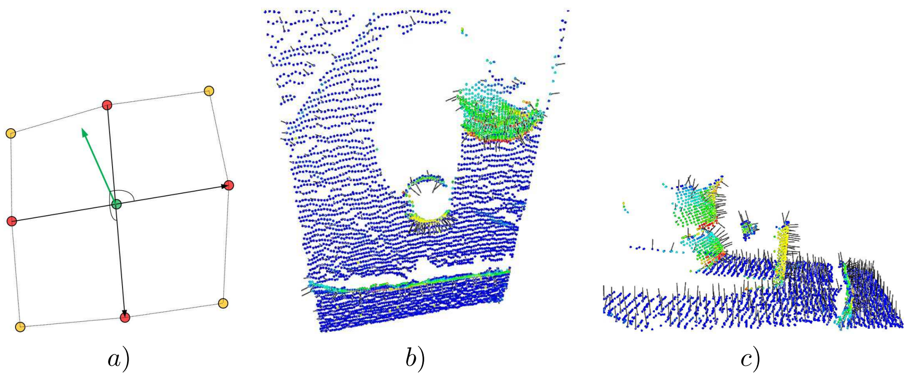


Figure 3.32: Local surface normal (LSN) feature calculation. Image $a)$ shows the LSN feature calculated for a green point, by taking a cross product of two tangent vectors obtained from the red points. Images $b)$ and $c)$ show a typical result of annotating the point cloud with LSNs (top and side views respectively). Taken from Holz et al. (2011).

In Hegger et al.'s (2013) approach, LSNs are used to obtain final human descriptors and to detect humans in the depth images in the following way. First of all, the input point cloud from RGB-D sensor is cropped to the region of interest ($0.5\,m \leq$ *depth* $\leq 5.0\,m$ and $0.0\,m \leq$ *height* $\leq 2.0\,m$). Secondly, the points in the ROI are subsampled using the voxel[6] size of 3 *cm* $\times$ 3 *cm* $\times$ 3 *cm*, and the LSNs are computed for all remaining points.

Afterwards, the remaining point cloud is segmented into horizontal slices (each of 25 *cm* height), and a basic Euclidean clustering technique is applied within each slice, *i.e.* two points are added the same cluster if the Euclidean distance between them is smaller than the threshold θ. (In their implementation, Hegger et al. set $\theta = 2 \times$ *cell size* $= 6$ *cm*.) Finally, all LSNs from the same cluster are added into a single-dimensional histogram, and this histogram, together with the cluster's width and depth, is used as a feature vector.

Hegger et al. train AdaBoost, SVM and Random Forest (Breiman, 2001) discriminative classifiers based on these histogram of LSN features, which are then used to classify unseen clusters into partial "human"/"non-human" objects. These part-based classifications are then merged using Euclidean clustering with the threshold θ set to $\theta = 2 \times$ *slice height* $= 50$ *cm*. Any connected component containing more than three clusters is considered as a detected human.

3.1.2.2 Other approaches for human detection from depth data

Human detection using template matching/model fitting Xia et al. (2011) describes an approach in which the humans are detected by first matching a two-dimensional head-and-shoulders template to the input depth intensity image to obtain the candidate head regions, and then verifying those regions by fitting a three-dimensional

[6]Volumetric pixel.

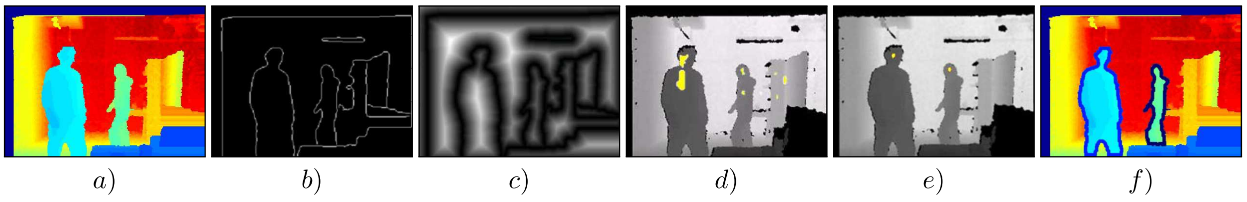


Figure 3.33: Human detection steps in Xia et al.'s (2011) method. Image *a*) shows the noise filtered depth image, image *b*) shows the edges found using Canny edge detector, image *c*) shows the distance transform (DT) of this image, the areas that satisfy the chamfer distance threshold between the DT image and the head-and-shoulders template (shown in figure 3.34) are coloured in yellow in image *d*), the regions in the image that satisfy the hemisphere fitting threshold (*i.e.* detected head centers) are shown in yellow in image *e*), and the human contours obtained using a region growing algorithm with detected head centers as seeds are shown in figure *f*). Adapted from Xia et al. (2011).

hemisphere to the depth image. After obtaining the head regions, the contours of the whole human body are extracted using a region growing method[7].

In particular, Xia et al.'s (2011) method works as follows:

1. Given the input depth array, the nearest neighbour interpolation method is used to fill the missing depth data (due to occlusions, depth shadows, out-of-range objects, *etc*). After removing the missing data, the 4×4 median filter is used to remove speckle and impulse noise.
2. Canny edge detector (Canny, 1986) is used to find the edges in the filtered depth array as illustrated in part *b*) of figure 3.33, and the multi-resolution edge image pyramid is generated from the original edge image using the subsampling rate of 0.75.
3. The distance transform (DT) for each of the edge images in the pyramid is calculated, as shown in part *c*) of figure 3.33.
4. A binary head-and-shoulders template shown in figure 3.34 is translated over the DT image pyramid; if the chamfer distance between the template and the DT image at a given location is smaller than a user-specified threshold, the location is marked as a candidate head center. This is illustrated in part *d*) of figure 3.33.
5. Assuming that a human head is present at each candidate location, the radius of the head r is approximated based on the candidate location's depth. This hypothesis of head's presence at a candidate location is then verified by extracting a circular part of the depth image (centred at the candidate location and with the approximated radius r), and calculating the square error between this circular part and a hemisphere model, shown in figure 3.35. If the square error exceeds another user-specified threshold, the candidate location is rejected. Image *e*) in figure 3.33 shows the final head center predictions.

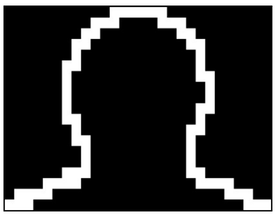


Figure 3.34: "Head-and-shoulders" template used for candidate head detection in the distance transform image. Taken from Xia et al. (2011).

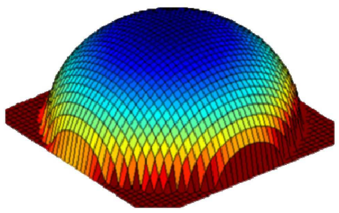


Figure 3.35: Hemisphere model used for candidate head verification. Taken from Xia et al. (2011).

[7]Growing the region until the depth difference between the neighbouring image locations exceeds a threshold.

6. A region growing algorithm is applied, using the detected head centres as seeds. This allows the whole body contours to be extracted, as shown in part f) of figure 3.33).

Human detection through clustering Basso et al. (2012) proposes a clustering-based approach for people detection in RGB-D data, which works in the following way:

1. First of all, the input point cloud is subsampled by dividing the input volume into three-dimensional grid of equal sized cells, and representing each cell with its centroid to improve the run-time performance of the algorithm, and to reduce the dependencies between the point cloud density and distance from the sensor (in Basso et al.'s approach the cell size is set to 6 *cm*). The subsampling process is illustrated in parts a) and b) of figure 3.36.

2. Secondly, the points on the ground plane are removed from the point cloud. This is achieved by asking the user to select three floor points in the RGB image, deriving the plane equation from these points and removing the points in the cloud which are within the threshold distance from the plane (during the initialization stage of the detector).

   In the subsequent stages, the equation of the ground plane is automatically refined by detecting all the points within a threshold from the hypothetical ground plane, and fitting a new plane using the least square distance optimization. The point cloud after ground plane removal is shown in part c) of figure 3.36.

3. After removing the ground plane, the remaining points are clustered based on their Euclidean distance. In particular, if the distance between two points is smaller than a pre-set threshold, they are placed into the same cluster.

4. The clusters are verified based on their geometrical properties: firstly, the clusters that contain less than 30 or more than 600 points are rejected, as shown in part d) of figure 3.36. Secondly, the clusters that have the height smaller than $1.4\,m$ or higher than $2.3\,m$ are rejected. Similarly, the clusters that are further than $6.5\,m$ away are rejected, since they would have too low point cloud density for reliable detection.

5. The remaining clusters (as shown in part e) of figure 3.36) are classified as being occluded or non-occluded, based on the positions of their bounding boxes (illustrated in figure 3.37). The human-likeness of non-occluded clusters is then examined by classifying the RGB image area corresponding to the cluster's bounding box with a SVM trained using HOG features. The clusters which are classified by the SVM as "non-human" are rejected, the remaining clusters are classified as human detections.

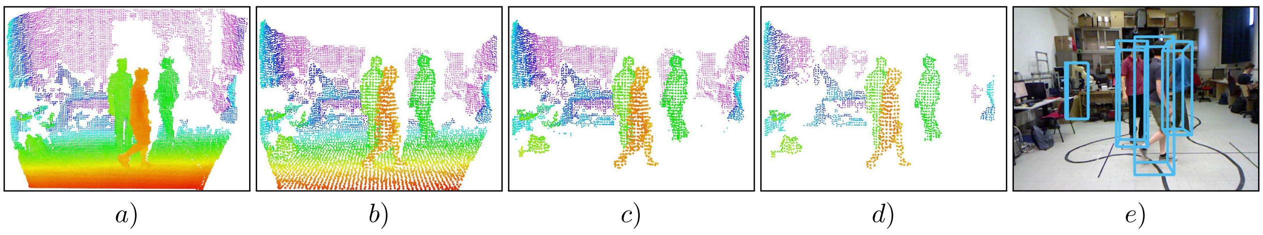


Figure 3.36: Human detection steps in Basso et al.'s (2013) method. Image a) shows the original point cloud, image b) shows the subsampled point cloud, image c) shows the point cloud after ground plane removal, image d) shows the clusters which contain between 30 and 600 points, and image e) shows the bounding boxes of the clusters that satisfy the height and distance constraints, overlaid over the original RGB input image. The clusters which are verified by the SVM trained on HOG features are considered as final human detections. Adapted from Basso (2011).

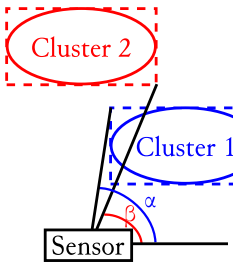


Figure 3.37: Occlusion evaluation criteria in Basso et al.'s (2013) method. Since $\alpha > \beta$, "Cluster 2" would be considered occluded by "Cluster 1". Adapted from Basso (2011).

Multiple detector fusion In the people detection and tracking approach described by Choi et al. (2011), multiple RGB-D data based detectors are combined into a probabilistic sampling-based framework in the following way.

Let $I_{1\sim t}$ be the sequence of RGB-D images in time instants $1, ..., t$, and $Z_t = \{Z_t^0, ..., Z_t^k\}$ be the set of human head locations at time t, with each Z_t^i given by a three-dimensional point (x, y, z). Using the sequential Bayes formulation (Arulampalam et al., 2002), the posterior probability $\Pr(Z_t|I_{1\sim t})$ can be expressed as

$$\begin{aligned}\Pr(Z_t|I_{1\sim t}) &= \frac{\Pr(I_t|Z_t, I_{1\sim t-1})\Pr(Z_t|I_{1\sim t-1})}{\Pr(I_t|I_{1\sim t-1})}\\ &\propto \Pr(I_t|Z_t)\Pr(Z_t|I_{1\sim t-1})\\ &= \Pr(I_t|Z_t)\int \Pr(Z_t|Z_{t-1})\Pr(Z_{t-1}|I_{1\sim t-1})dZ_{t-1} \text{ \textit{(Using Chapman-Kolmogorov eq.)}}\\ &= \prod_i \Pr(I_t|Z_t^i)\int \prod_i \Pr(Z_t^i|Z_{t-1}^i)\Pr(Z_{t-1}|I_{1\sim t-1})dZ_{t-1}. \text{ \textit{(Using target independence)}}\end{aligned}$$

The motion prior $\Pr(Z_t^i|Z_{t-1}^i)$ for each target is modelled as a Gaussian distribution centred over Z_{t-1} (to ensure motion smoothness), together with two binomial probabilities representing the possibilities of new target's appearance, and the likelihood of target's persistence between time instants t and $t-1$. The observation likelihood is modelled using $\Pr(I_t|Z_t^i) \propto \exp\left(\sum_j w_j l_j(I_t|Z_t^i)\right)$, where $l_j(\cdot)$ is an individual detector log likelihood.

In Choi et al.'s approach the following individual detectors are used:

1. Upper body and full body HOG detectors (l_{HOG}), as described earlier,
2. Viola-Jones face detector (l_{face}), as described earlier,
3. Shape detector (l_{shape}), which is based on the Hamming distance between the two-dimensional binary "head-and-shoulders" shape and thresholded depth image in the target area,
4. Skin detector (l_{skin}), which checks what proportion of the pixels in the target area are within a "skin colour" region in HSV colour space,
5. Motion detector (l_{motion}), which works by identifying the differences between the point clouds in time instants t and $t-1$, projecting those points back into the image, and calculating the proportion of moving pixels in the target area.

Since the posterior at $t-1$ time instant ($\Pr(Z_{t-1}|I_{1\sim t-1})$) is not tractable to be calculated exactly, the resulting posterior probability at time t is approximated as

$$\Pr(Z_t|I_{1\sim t}) \propto \prod_i \Pr(I_t|Z_t^i) \sum_{r=1}^{N} \Pr(Z_t|Z_{t-1}^{(r)}),$$

where target position sets $\{Z_{t-1}^{(r)}\}_{r=1}^{N}$ are sampled from $\Pr(Z_{t-1}|I_{1\sim t-1})$ distribution using reversible-jump Monte Carlo Markov Chain (RJ-MCMC, Khan et al. (2004)) method (see Choi et al. for full details on the target proposal Markov Chain). This approach also enables tracking-by-detection.

To start the detection/tracking process, the initial target location set Z_0 is approximated by removing the floor plane from the initial point cloud (using the extrinsics of RGB-D sensor *w.r.t.* robot), clustering the remaining points and using the highest points from each cluster as head location proposals. Only those proposals which are between 1.3 and 2.3 metre distance from the ground, are used as actual initialization parameters.

3.2 Obstacle avoidance

An autonomous robot photographer, just like any other autonomous mobile robot, needs to be able to navigate in its environment without colliding with any obstacles that it encounters. Traditional sensors used for this task include laser range, ultrasound, infrared sensors, stereo or monocular cameras, or various combinations of any of the above. Each of those sensors have their individual limitations, as summarized by in table 3.1.

Sensor	Speed	Cost	Power	Density	Accuracy	Dimensions
Infrared sensor ring	Fast	Low	Low	Low	Low	2
Ultrasound sensor ring	Fast	Medium	Low	Low	Medium	2
Laser range finder	Slow	High	High	High	High	2
LIDAR	Slow	High	High	High	High	3
Single camera	Fast	Low	Low	High	Low	3
Multiple cameras	Fast	Low	Low	Low	Medium	3
RGB-D sensor	Fast	Low	Low	High	Medium	3

Table 3.1: A comparison of different sensors used for obstacle detection. Extended from Peasley and Birchfield (2013).

Infrared or ultrasonic sensors arranged in a ring-like formation around the robot can provide planar range readings at a low cost, but the acquired distance values are sparse and not very accurate. Laser range finders can produce much more accurate and dense planar range readings, but they consume a large amount of power and cost thousands of dollars. When combined with spinning mirror systems, laser range finders can also provide three dimensional range scans (so-called LIDAR systems), but at the expense of decreased frame rate. In image-based obstacle avoidance systems, the distances have to be inferred from the image data, requiring either multi-camera rigs or robot's movement (exploiting the motion parallax). This requires solving the computationally-intensive correspondence problems (spatial or temporal), and typically produces only sparse, image noise sensitive distance measures.

The active-light based RGB-D sensors overcome most of these limitations, by providing real-time, dense, three dimensional distance readings at affordable prices[8]. Since these sensors can be used with both image- and depth-based obstacle avoidance techniques, both types of methods are briefly surveyed below.

[8]Microsoft Kinect RGB-D sensor costs under one hundred dollars at a time of writing this thesis.

3.2.1 Survey of image-based obstacle avoidance methods

A large number of image-based methods have been proposed for mobile robot navigation and obstacle avoidance (*e.g.* see surveys by Bonin-Font et al. (2008) or DeSouza and Kak (2002)). Notable image-based approaches, which do not rely on a prior map of the environment include the *optical flow* techniques, image *qualitative characteristic extraction* methods and *feature tracking* based approaches (using the taxonomy of Bonin-Font et al.), each of which are described below.

Optical flow based methods Methods that rely on the optical flow try to detect the ground plane and the obstacles above it from the motion of spatial features in a sequence of images.

An example method for obstacle detection based on optical flow is described by Santos-Victor and Sandini (1995). In their method they assume that the camera is facing the flat ground surface (on which the robot is moving) defined by the equation $Z(x,y) = \frac{Z_0}{1-\gamma_x \frac{x}{f_x} - \gamma_y \frac{y}{f_y}}$, where f_x, f_y are the camera's horizontal and vertical focal lengths and Z_0, γ_x, γ_y are the parameters of the plane (the distance along the optical axis, and slant/tilt respectively). Then, from the image brightness constancy equation for the optical flow ($u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} = -\frac{\partial I}{\partial t}$, where $u \approx \frac{\partial x}{\partial t}, v \approx \frac{\partial y}{\partial t}$) they derive the normal flow equation for the planar surface in motion as

$$\left[\frac{\partial I}{\partial y},\ x\frac{\partial I}{\partial y},\ y\frac{\partial I}{\partial y},\ \frac{\partial I}{\partial x},\ x\frac{\partial I}{\partial x},\ y\frac{\partial I}{\partial x}\right] \boldsymbol{\theta} = -\frac{\partial I}{\partial t},$$

where the parameter vector $\boldsymbol{\theta} = [v_0,\ v_x,\ v_y,\ u_0,\ u_x,\ u_y]^T$ is estimated using a recursive least squares approach. Using the obtained parameters $\boldsymbol{\theta}$, the equation of the ground plane can be recovered (up to a scale factor) by calculating $\gamma_x = -\frac{v_x}{v_0} f_x$ and $\gamma_y = -\frac{u_y}{u_0} f_y$. (Santos-Victor and Sandini also provide a way to estimate the equation of the ground plane even if the intrinsic parameters of the camera f_x, f_y are not known.)

Finally, given the equation of the ground plane the obstacles can be detected by observing that for a translational motion, the points on the ground surface should have the same flow vectors (after re-projecting the flow field onto this estimated ground plane). Points above or below the ground should then have different magnitude or orientation flow vectors, and thus can be identified as obstacles, as illustrated in figure 3.38.

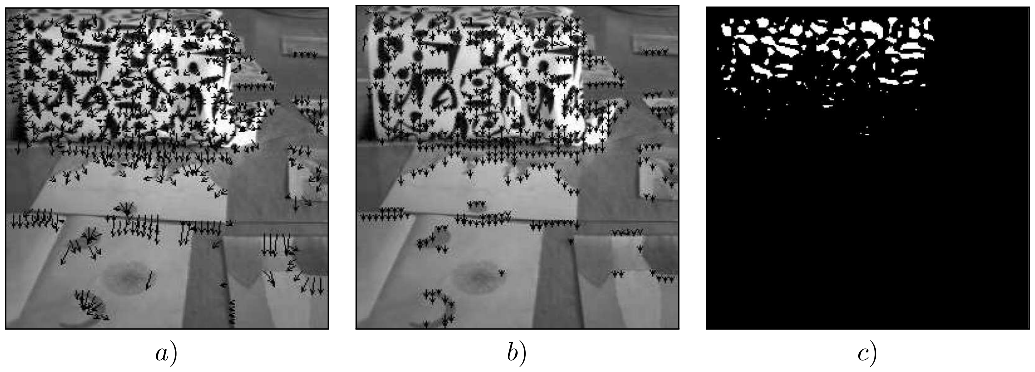


Figure 3.38: Optical flow-based obstacle detection: image a) shows the normal flow field, image b) shows the flow field re-projected onto the ground plane, image c) shows the detected obstacles by looking for variations in otherwise constant flow field. Adapted from Santos-Victor and Sandini (1995).

In a more recent approach by Braillon et al. (2006), optical flow based obstacle detection is combined with stereo obstacle detection. For obstacle detection from the stereo information, a calibrated two-camera rig is pointed towards the ground plane, and the 3D point cloud obtained from the stereo image pairs is used to estimate the ground plane equation by fitting a plane to the obtained point cloud using the Least Median Squares method (under the same flat ground assumption). Then the occupancy grid $C_s(x, y)$ (containing obstacles and free space, as illustrated in part c) of figure 3.39) can be generated by thresholding the shortest distance between each point in the obtained cloud and the estimated ground plane.

For the obstacle detection using optical flow (based on the image input from one of the cameras in the stereo pair), an odometry-based approach is used. In particular, the theoretical optical flow is generated, which should be observed on an estimated ground plane for a given linear and angular velocity of the robot. Then this generated optical flow field is matched with the actual optical flow obtained between two consecutive image input frames I_t and I_{t+1}, calculating the image of squares of differences between the model and the observed flow $I_\triangle$, as illustrated in part b) of figure 3.39. The discrepancies between the observations and the model are assumed to have occurred due to the obstacles in robot's path.

Due to the fact that a single pixel in the image could have been generated by any three-dimensional real-world point on the pixel's projective line, image $I_\triangle$ actually generates a pyramid $C_o(x, y, z)$ of occupied points in 3D space, constructed by projecting each point in $I_\triangle$ onto the ground plane estimated from the stereo data. To collapse this pyramid into the two dimensional occupancy grid $C_o(x, y)$, Braillon et al. use the following probabilistic formulation:

$$C_o(x, y) = \max_z \left(\Pr(z) C_o(x, y, z) + (1 - \Pr(z)) \frac{1}{2} \right),$$

where $\Pr(z)$ represents a prior probability for an obstacle observed at height z to extend all the way to the ground floor (constructed to decay with increasing height). After this projection, the occupancy map obtained from $I_\triangle$ is shown in part d) of figure 3.39. Finally, the obtained occupancy maps from stereo and optical flow data are merged, yielding the final occupancy map illustrated in part e) of figure 3.39.

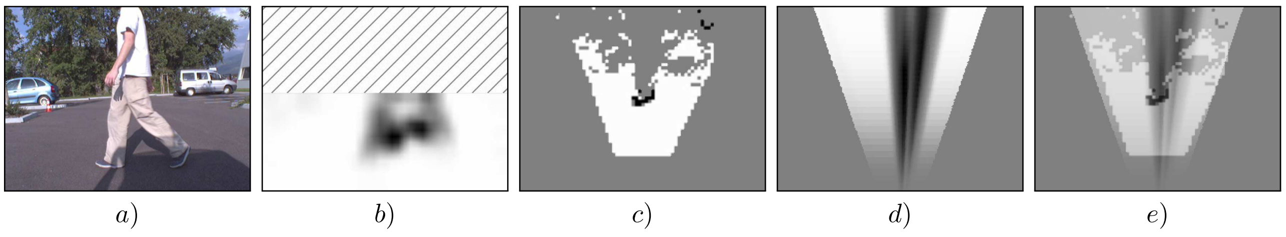


Figure 3.39: Obstacle detection from optical flow and stereo data in Braillon et al.'s (2006) method. Image a) shows the original input image for one of the cameras in the stereo pair, image b) shows the square of differences image $I_\triangle$ between the odometry-based optical flow model and the actual optical flow obtained from two input frames (ignoring differences above the horizon line), image c) shows the occupancy grid $C_s(x, y)$ calculated from the point cloud obtained from the stereo input data, image d) shows $I_\triangle$ projected and collapsed into the occupancy grid $C_o(x, y)$ (as described in the text), and image e) shows the merged occupancy grids C_s and C_o. Adapted from Braillon et al. (2006).

Qualitative feature extraction In contrast to quantitative approaches, which try to calculate accurate numerical data like distances to obstacles or their coordinates in the world plane, the qualitative approaches try to extract characteristic features from the visual data in order to distinguish between free and occupied space.

An early example of such system is described by Lorigo et al. (1997). Their approach is based on two assumptions: that the ground plane is flat, and that the boundaries between the obstacles and the ground surface are visible in the image. Then Lorigo et al. use detected edges and normalized RGB/HSV histograms to detect obstacle boundaries in the following way.

First of all, the input image is scanned from the bottom towards the top in vertical slices, assembling edge/RG-B/HSV feature histograms in each window. The edge feature histogram is calculated by simply assembling the gradient magnitudes in a given window into a histogram. Similarly, the RGB/HSV-based feature histograms are built from the red and green colour intensities/hue and saturation values within the window. The lowest window in each slice is assumed to be obstacle free; then all higher windows which have sufficiently different feature histograms are assumed to contain obstacles (as illustrated in figure 3.40).

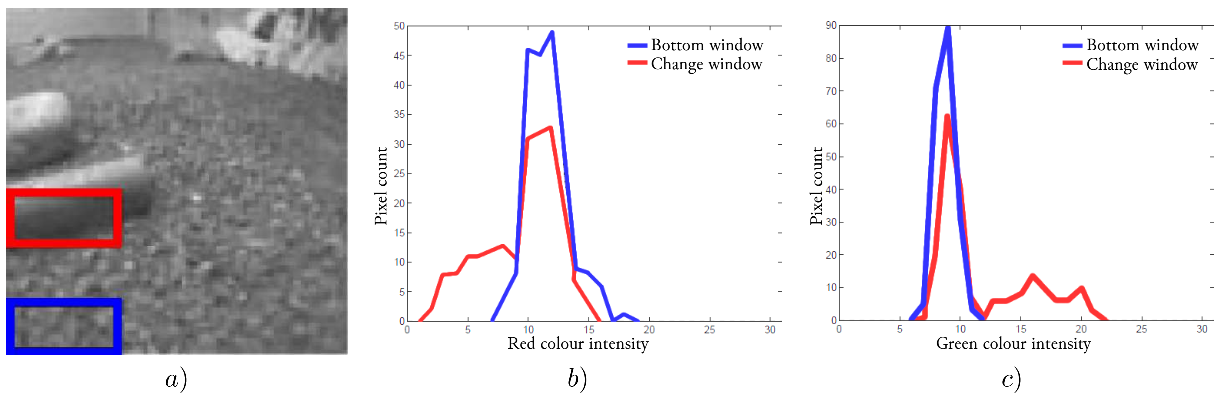


Figure 3.40: Histogram-based obstacle detection in Lorigo et al.'s (1997) method. Image $a)$ shows the 64×64 pixel size input for the algorithm together with two windows, the lowest of which is assumed to be obstacle free. Images $b)$ and $c)$ show the 32-bin histograms of red and green pixel intensities under each window; the window would be declared to contain obstacles if the cumulative difference between the histograms exceeds a pre-set threshold. Adapted from Lorigo et al. (1997).

Another qualitative feature extraction based method is described by Maja et al. (2000). Their approach (based on the same assumptions) is even simpler. First of all, an input grayscale image I is binarized using $I(x, y) > \mu - \sigma$ threshold, where μ is the arithmetic mean and σ is the standard deviation of the intensities in the image. Then the binarized image $\tilde{I}(x, y)$ is scanned bottom-to-top using one pixel width vertical stripes, and the earliest occurrence of the pixel's intensity under the threshold is recorded for each stripe (*i.e.* $H(x) = \min_y\{y \mid \tilde{I}(x, y) = 0\}$).

The robot chooses the direction for navigation by calculating the largest free-space area in the binarized image, *i.e.* the direction corresponding to the horizontal coordinate $\hat{x} = \arg\max_x \sum_{x'=x-\delta/2}^{x+\delta/2} H(x')$, where δ is the robot's width in pixels.

Feature tracking approaches In the feature tracking approaches the robust elements of the image (lines, corners, object outlines) are used to estimate the location of the ground plane in the input image from the homographies between feature points.

An example obstacle avoidance method based on feature tracking is described by Pears and Bojian (2001). In their approach, Harris and Stephens (1988) corner detector is used to extract the corner points in the image. These points are tracked for n frames using a Kalman filter, and a homography (plane-to-plane projection) H is calculated between a limited set of corner points from frames 1 and n. The calculated homography H can then be used to check if other corner points lie on the same plane; the homography that verifies the largest number of corner associations is assumed to be mapping corner points between the ground planes in both frames. This allows the corner points on the ground plane (verified using H) to be grouped together in patches, and the colour model of the ground plane to be extracted from the image area under those patches. At this point, the whole image can be classified into "ground plane"/"non ground plane" regions based on their colour, as illustrated in figure 3.41.

After segmenting the image into "ground plane"/"non ground plane" regions, Pears and Bojian apply a modified potential field method (Borenstein and Koren, 1989) to generate a fictional force that provides the direction and speed instructions for the obstacle avoiding robot. In particular, they cast imaginary rays from the bottom center of the image to the edges of the segmented ground plane region, and generate the force vectors in the opposite directions (*i.e.* towards the bottom center of the image), which have the magnitudes inversely proportional to the

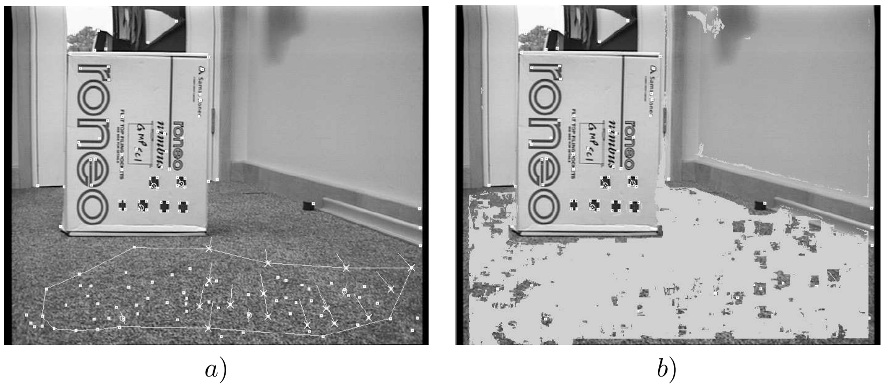


Figure 3.41: Ground plane segmentation in Pears and Bojian's (2001) method, based on calculation of the homography that relates ground planes in temporally separated frames. Image a) shows the extracted corner points on the ground plane grouped into a patch, image b) shows the extracted ground plane region, based on its colour similarity to the colour of the patch in image a). Adapted from Pears and Bojian (2001).

length of the cast rays. Adding these force vectors together and including a forward-pointing driving force produces the resulting direction vector.

A number of similar systems based on homography calculation from the feature points have also been proposed (*e.g.* see the descriptions by Dao et al. (2005) or Zhou and Li (2006)).

3.2.2 Survey of depth-based obstacle avoidance methods

Approaches to depth based obstacle avoidance in unstructured environment can be split into two categories, based on their need for the structural assumption of the sensor's position. Methods in the first category use the structural knowledge about the sensor's position/tilt angle (with respect to the rest of the robot) to detect the ground plane in the input data. Methods that do not make this assumption detect the ground plane using only the input from the RGB-D sensor.

An example of the latter approach (involving no prior structural knowledge about the sensor's position) was proposed by Nguyen (2012). In particular, Nguyen describes a method that detects the ground plane solely from the depth input in the following way. First of all, the input point cloud is subsampled using the voxel filter (as described in section 3.1.2), and the points that are closer than $0.5\,m$ or further than $1.4\,m$ are eliminated.

The remaining points are used as inputs for the Random Sample Consensus (RANSAC, Fischler and Bolles (1981))

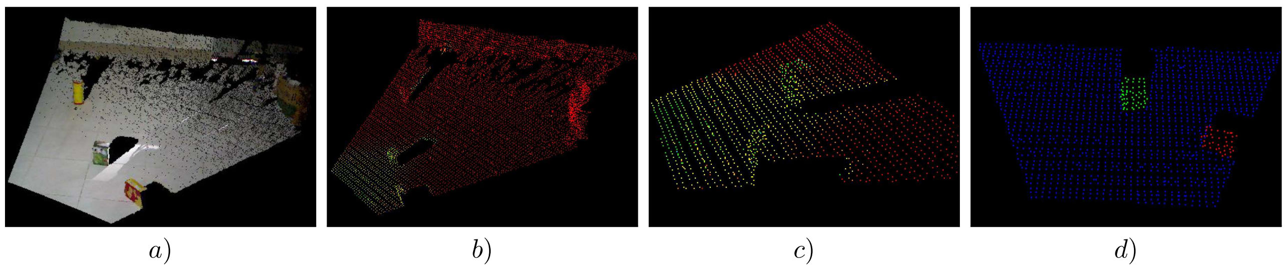


Figure 3.42: Obstacle detection process in the approach by Nguyen (2012). Image a) shows the original input RGB point cloud, image b) shows the point cloud after the voxel filtering, and image c) shows the depth cloud after removing points with distances $> 1.4\,m$ or $< 0.5\,m$. Image d) shows the final results after the ground plane detection using RANSAC algorithm (shown in blue) and obstacle clustering based on their Euclidean distance (shown as green and red clusters). Adapted from Nguyen (2012).

algorithm, which is used to find the equation of the dominant plane (assumed to be the ground plane). Essentially, this procedure works by sampling three non-collinear points from the remaining point cloud (which uniquely define an equation of the plane), counting the number of points further than a threshold distance from the defined plane and repeating the process if this number is too large (or for a fixed number of iterations).

When the RANSAC process terminates, the points belonging to the floor plane are discarded and the remaining points are grouped using Euclidean clustering; the obtained clusters are returned as detected obstacles. This process is illustrated in figure 3.42.

However, the majority of obstacle detection in RGB-D data methods use *a priori* knowledge about the position and pitch of the sensor with respect to the robot's base. Examples of such methods are presented by Mojtahedzadeh (2011), Holz et al. (2011) and Peasley and Birchfield (2013), each of which is briefly described below.

In Mojtahedzadeh's (2011) approach, the prior knowledge about the sensor's position and pitch angle is exploited by rotating and translating the input point cloud to the robot's coordinate frame. Then all the points with the heights outside the minimum and the maximum height boundaries of the robot are removed, while the remaining points are projected into a 2D obstacle map, as illustrated in figure 3.43.

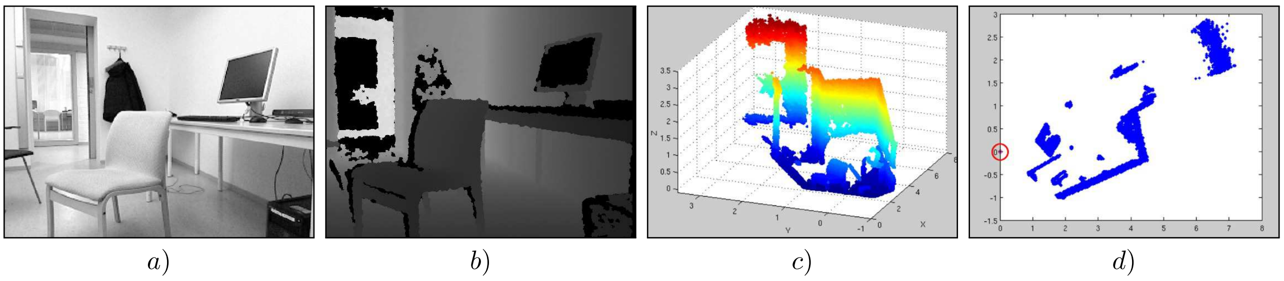


Figure 3.43: Obstacle detection process in Mojtahedzadeh's (2011) approach. Images a) and b) show the original RGB/depth input images, image c) shows the obtained point cloud, and image d) shows the obstacle map, obtained by removing the points outside the robot's height (including the floor plane) and projecting the remaining points vertically into two dimensions. The red circle in image d) shows the robot's location. Adapted from Mojtahedzadeh (2011).

In the alternative Holz et al.'s (2012) approach, the local surface normals are first calculated from the input point cloud (as described in section 3.1.2. Then the calculated surface normals are transformed into the base coordinate frame of the robot, using the knowledge about the sensor's position *w.r.t.* to the robot's base.

Afterwards, the transformed points are clustered based on their (transformed) normals $\boldsymbol{n} = [n_x, n_y, n_z]^T$ by first putting all the points with normals $|n_x - x| < \delta$, $|n_y - y| < \delta$, $|n_z - z| < \delta$ into the same cluster $C_{(x,y,z)}$, and then merging the neighbouring clusters (with similar normal orientations) until sufficiently large clusters are obtained. The resulting clusters represent the sets of planes with similar surface orientation, but the planes themselves might be spatially separated (*e.g.* see the red cluster in part b) of figure 3.44 composed of multiple planes).

To mitigate this problem, the average normal vectors $\overline{\boldsymbol{n}}_i$ are calculated for each cluster C_i, and the distances from the origin to a plane defined by $\overline{\boldsymbol{n}}_i$ and each point $\boldsymbol{x}_i \in C_i$ are calculated. Clearly, this distance is the same for the points on the same plane, hence the histograms of these distances are assembled and points from the same bin are put into the same cluster. After obtaining the final clusters, the points in a horizontal plane $n_z \approx 1$ with $z \approx 0$ are considered as obstacle-free points on the ground plane (shown in gray in part c) of figure 3.44).

In another example of obstacle detection from depth data, Peasley and Birchfield (2013) use an identical approach to Mojtahedzadeh (2011) to obtain the initial obstacle map. However, to further improve the obstacle detection, Peasley and Birchfield (2013) propose a way to overcome the limitations of Kinect in high specularity parts of the scene.

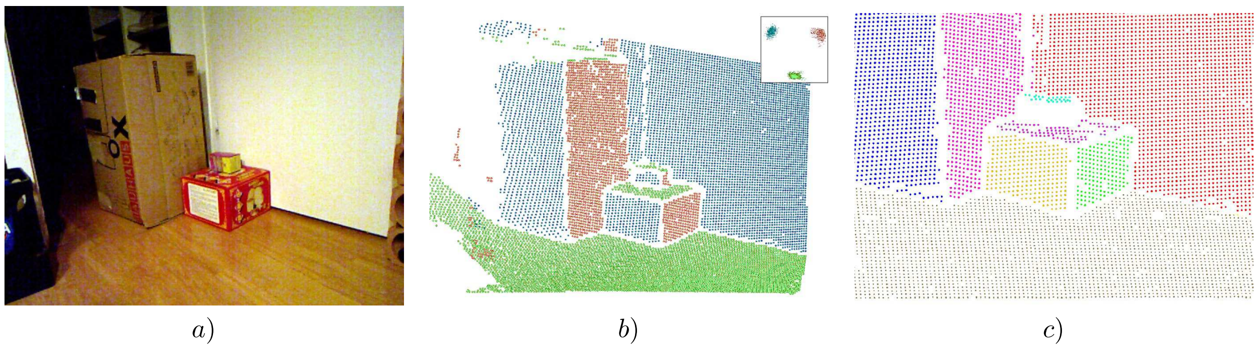


Figure 3.44: Obstacle detection based on point cloud clustering in Holz et al.'s (2012) approach. Image a) shows the original RGB input, image b) shows three clusters (red, green and blue) obtained after clustering the input points based on their local surface normals, and image c) shows the eight final plane clusters obtained after point re-clustering based on the plane-to-origin distance, as explained in the text. The obtained obstacle-free ground plane segment is shown in gray. Adapted from Holz et al. (2011).

In particular, given a pixel with an invalid depth reading, its neighbours in the depth image are examined. If any of the neighbours lie on the floor plane (approximated as points with $z < 5\,cm$) then that floor point is marked as an obstacle in the 2D obstacle map (which semantically corresponds to placing an infinitely tall obstacle at that floor point). This improvement is illustrated in figure 3.45.

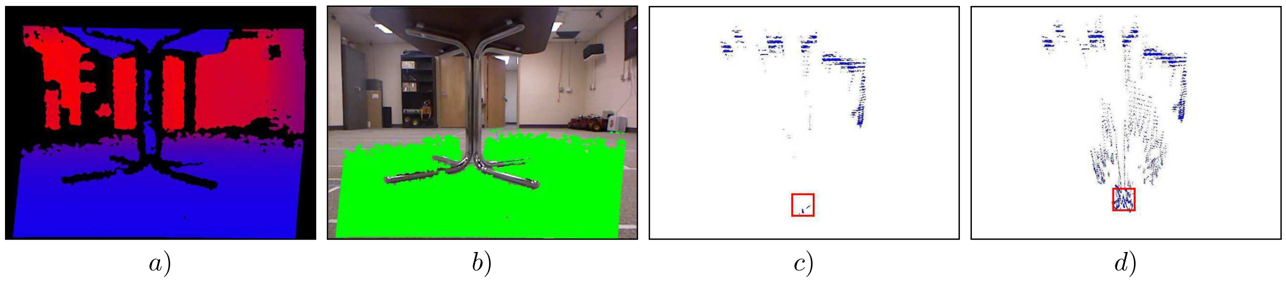


Figure 3.45: Improvements to reflective object detection by Kinect. Image a) shows the original depth input, image b) shows the RBG input with the floor segment coloured in green. Notice that the table with metallic surface have no depth readings in image a) (coloured in black). Image c) shows the obstacle map obtained by ignoring invalid depth readings (with the table area marked as a red rectangle), while image d) shows the obstacle map obtained after specular surface marking as described in text. Adapted from Peasley and Birchfield (2013).

Another improvement proposed by Peasley and Birchfield relates to the loss of depth data when the objects are too close to the sensor (this can occur due to moving objects in the scene that suddenly appear in front of the sensor, or when the robot turns away from one obstacle and starts facing another). This problem is mitigated by examining the proportion of pixels in the input depth image that do not have valid depth readings. If this proportion exceeds 40% then the robot proposed by Peasley and Birchfield starts turning continuously in-place until the proportion of valid pixels increases.

3.3 Proposed methods for the use in an autonomous robot photographer

The human detection/tracking and obstacle detection/avoidance methods for the use by an autonomous robot photographer are selected mostly based on their computational efficiency. This condition is imposed by the simplicity of the on-board computer in the robot (Intel Atom N2800 1.6 GHz CPU, 1 GB RAM, no dedicated GPU) and power constraints (the use of complicated algorithms requires more power, which quicker drains the mobile robot's battery).

The feasibility for a given method to be implemented within the project's timeframe is also taken into consideration, thereby rejecting methods that require training datasets which are not publicly available, or need training processes which are time or computing resources intensive.

The use of these criteria is strengthened by the assumption that even simple RGB-D human/obstacle detection algorithms should outperform similar image-based algorithms due to the richness of RGB-D data, hence by providing a highly modular and decoupled implementation of the robot's control system the incorporation of more sophisticated human detection algorithms could be left for the future research.

3.3.1 Proposed human subject detection method

With the above considerations in mind, a knowledge-based head detection algorithm by Garstka and Peters (2011) is chosen and extended to cope with multiple people presence in the image. To improve the head detection results, two skin detectors are implemented: a Bayesian skin detector by Jones and Rehg (2002), and an adaptive skin detector based on a logistic regression classifier with a Gaussian kernel, and trained on an on-line skin model obtained from the face regions detected using Viola and Jones (2001) detector. Finally, to exploit the spatial locality of human heads over a sequence of frames, a depth-based extension of the continuously-adaptive mean-shift algorithm by Bradski (1998) is proposed. Each of these methods are further explained below, starting with the head localization algorithm of Garstka and Peters (2011).

3.3.1.1 Head localization from depth images using Garstka and Peters (2011) approach

Garstka and Peters (2011) describe a fast, knowledge-based human head localization[9] method. It consists of three main steps: *i*) depth shadow elimination, *ii*) depth image smoothing, and *iii*) head localization through local minima detection and verification of surrounding geometrical features, based on prior knowledge about human head sizes. These steps are described in more detail below.

Depth shadow elimination Due to the fact that IR projector is placed 2.5 *cm* to the right of the IR camera in Kinect sensor, the depth shadows (places visible by IR camera which do not have a projected IR pattern) always appear on the left side of a convex object (as illustrated in figure 3.46).

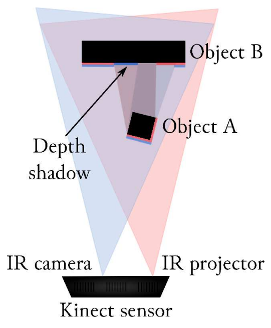


Figure 3.46: Kinect depth shadows for the convex objects. Light blue polygon shows the area visible from the IR camera's point of view, light red polygon shows the projected IR pattern. Blue lines indicate the areas on the object surfaces visible by the IR camera, red lines indicate the areas which have a projected IR pattern. Note that depth shadows always appear on the left side of convex objects. Adapted from Zabarauskas (2012).

[9]Borrowing the wording from Yang et al. (2002), head localization task can be defined as the head detection task under the assumption that an input image contains exactly one head. In other words, it is the task of determining the location of exactly one viewer's head in the input image.

Since human heads are indeed convex, Garstka and Peters propose to eliminate the depth shadows by scanning the image one horizontal line at a time, top-to-bottom, and replacing every unknown depth value with the last known one. This process is illustrated in part *c*) of figure 3.48.

Depth image smoothing Depth images can contain noise due to inaccurate measure of disparities in the correlation algorithm, external IR radiation (*e.g.* sunlight), high specularity object surfaces and so on. Since the subsequent detection step will involve treating every local horizontal minimum point as a potential point on a vertical head axis, it is highly prone to noise. To mitigate this problem, Garstka and Peters propose to use the integral image representation for fast smoothing. In particular, given the integral image $\tilde{I}$ and the smoothing radius r, the smoothed depth value $I_r(x, y)$ can be calculated using

$$I_r(x,y) = \frac{\tilde{I}(x+r, y+r) - \tilde{I}(x-r, y+r) - \tilde{I}(x+r, y-r) + \tilde{I}(x-r, y-r)}{(2r+1)^2}. \tag{3.6}$$

The input depth images blurred using radii $r \in \{2, 4, 8\}$ are shown in figure 3.47.

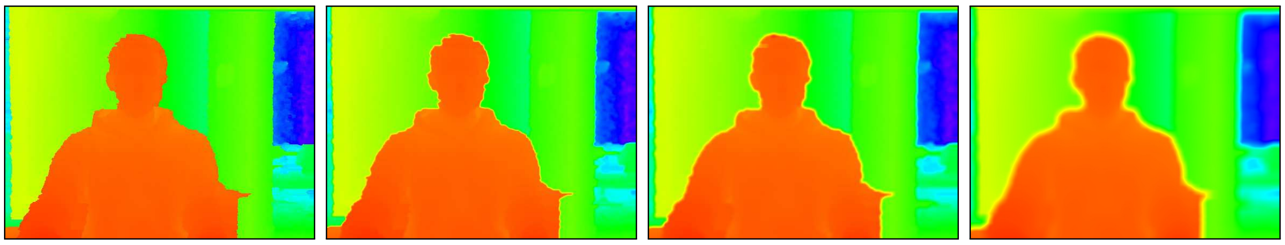


Figure 3.47: Depth image blurring using integral image approach.

Head localization In order to efficiently localize the head, Garstka and Peters start by making an assumption that the adult head has an approximate size of $20\,cm \times 15\,cm \times 25\,cm$ $(D \times W \times H)$. Since the exact head's orientation is unclear, Garstka and Peters assume the inner depth and width bound of $10\,cm$ and the outer depth and width bound of $25\,cm$.

Given an object with width w and height h, at a distance d from the Kinect sensor, the width and height that the object occupies on the screen ($p_w(d)$ and $p_h(d)$ respectively) can be calculated using

$$(p_w(d), p_h(d)) = \left(\frac{w \times r_w}{d \times 2 \tan \frac{f_w}{2}}, \frac{h \times r_h}{d \times 2 \tan \frac{f_h}{2}} \right), \tag{3.7}$$

where (f_w, f_h) are the horizontal/vertical fields-of-view (FOV) of the depth camera, and $r_w \times r_h$ is the resolution of the depth image. Garstka and Peters empirically measure a horizontal FOV of $61.7°$, which nearly corresponds to the PrimeSense PS1080 SoC reference design 1.081, which states $58°$ horizontal and $45°$ vertical FOVs.

Using equation 3.7, the inner and outer bound pixel distances at distance d (in a VGA resolution depth image) can be expressed as

$$b_i(d) = \frac{10\,cm \times 640\,px}{d \times 2 \tan \frac{58°}{2}} \approx \frac{5,773}{d} px,$$

$$b_o(d) = \frac{25\,cm \times 640\,px}{d \times 2 \tan \frac{58°}{2}} \approx \frac{14,432}{d} px.$$

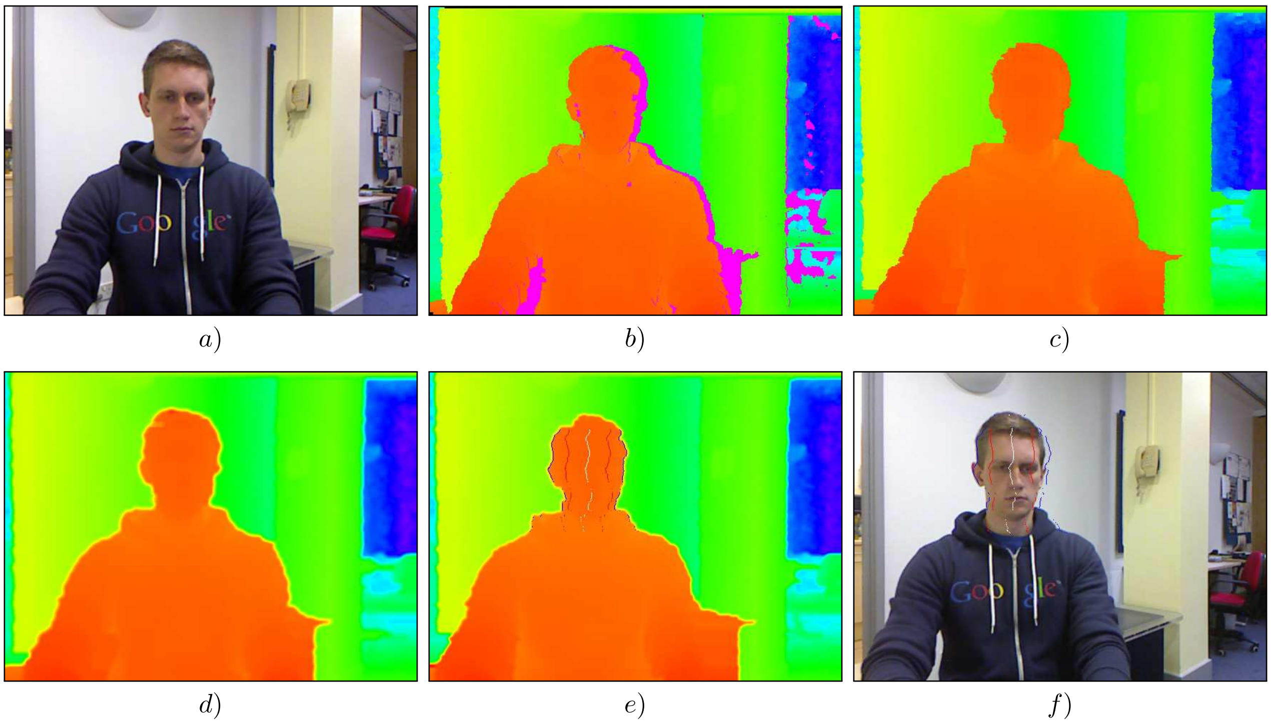


Figure 3.48: Head detection steps in Garstka and Peters' (2011) method. Images $a)$ and $b)$ show the original RGB/depth images obtained from Kinect sensor, with missing depth data annotated in purple, image $c)$ shows the depth image after filtering depth shadows, image $d)$ shows the result of depth blurring using the blur filter size $r = 4$, and images $e)$,$f)$ illustrate the detected head's inner bounds (shown in red), outer bounds (shown in blue) and the vertical head axis (shown in white).

Given these inner/outer bound pixel distances, the head location can be determined in the following way:

1. The depth image is scanned top-to-bottom, one horizontal line at a time.

2. For each horizontal line y, a local minimum pixel x is found such that the depth differences between this pixel and the depth values within the inner bounds are smaller than 10 *cm*, and the depth differences between this pixel and the depth values at the outer bounds are larger than 20 *cm*.

   More formally, x has to satisfy the following conditions:

$$\begin{gathered} d(x,y) < d(x \pm 1, y), \\ \forall \Delta_x \in \left\{1, \dots, \frac{b_i(d(x,y))}{2}\right\} : d(x \pm \Delta_x, y) - d(x,y) < 10\,cm, \\ d(x \pm \frac{b_o(d(x,y))}{2}, y) - d(x,y) > 20\,cm. \end{gathered} \tag{3.8}$$

3. In order to obtain accurate horizontal boundaries for the candidate head, the positions where the depth difference exceeds 20 *cm* are found on each side of x (named the left/right lateral gradients of x and denoted x_l and x_r respectively).

   In particular, x_l and x_r must satisfy the following constraints:

$$\begin{gathered} \forall g \in \{l, r\} : d(x_g, y) - d(x,y) \leq 20\,cm, \\ d(x_l - 1, y) - d(x,y) > 20\,cm, \\ d(x_r + 1, y) - d(x,y) > 20\,cm. \end{gathered} \tag{3.9}$$

Then the arithmetic mean of left and right lateral gradients $\overline{x}(y) = \frac{x_l(y)+x_r(y)}{2}$ is stored as a potential point on the vertical head axis (illustrated in parts e) and f) of figure 3.48).

4. Since Garstka and Peters assume the head height of at least 25 *cm*, a number of subsequent lines must satisfy the constraints from equation (3.8). The average depth of the points found in the last n lines can be calculated as

$$\overline{d} = \frac{1}{n}\sum_{i=0}^{n-1} d(\overline{x}(y-i), y-i).$$

Then the required number of lines satisfying the equation (3.8) can be obtained from equation (3.7) as

$$n_{max} = \frac{25, cm \times 480\,px}{\overline{d} \times 2\tan\frac{45^\circ}{2}} \approx \frac{14,485}{\overline{d}}.$$

5. If at least n_{max} lines satisfying the equation (3.8) are found while scanning the horizontal line y, then the head detection is triggered and the position of the head center $(\tilde{x}, \tilde{y})$ is returned, where

$$(\tilde{x}, \tilde{y}) = \left(\frac{1}{n_{max}}\sum_{i=0}^{n_{max}-1} \overline{x}(y-i),\ \ y - \frac{n_{max}}{2}\right).$$

3.3.1.2 Multiple people detection using an extension of Garstka and Peters (2011) method

Since a photographer robot needs to be able to detect multiple people in its environment at the same time, an extension to Garstka and Peters' (2011) method is proposed to cope with multiple people detection.

The extended method still scans through a blurred and depth-shadow-filtered depth image one horizontal line at a time, from top to bottom. However, instead of keeping a single potential vertical head axis, a set of vertical head axes $\{H_1, ..., H_k\}$ is constructed. Each head axis is represented by $H_i = \left(\overline{d}_i, (X,Y)_i\right)$, where $\overline{d}_i$ is the average candidate head distance from the sensor, and $(x,y) \in (X,Y)_i$ are the image points on the vertical axis.

When a new arithmetic mean of left and right lateral gradients $\overline{x}(y)$ is calculated (step 3 of the original algorithm), the extended method searches for the head axis H_j *s.t.* the last added point $(x', y') \in (X,Y)_j$ is within 5 *cm* distance from the point $(\overline{x}, y)$.

More specifically, let $w_d(p)$ and $h_d(p)$ be the width and height in centimetres of an object which occupies p pixels in the image while being at a distance d from the sensor. Inverting the equation (3.7), these measures can be expressed as

$$(w_d(p), h_d(p)) = \left(\frac{p \times d \times 2\tan\frac{f_w}{2}}{r_w}, \frac{p \times d \times 2\tan\frac{f_h}{2}}{r_h}\right). \tag{3.10}$$

Furthermore, let d be the depth of pixel $(\overline{x}(y), y)$. Then the 5 *cm* distance constraint described above can be expressed as

$$\sqrt{\left(w_d(\overline{x}(y)) - w_{\overline{d}_j}(x')\right)^2 + \left(h_d(y) - h_{\overline{d}_j}(y')\right)^2 + \left(d - \overline{d}_j\right)^2} < 5\,cm. \tag{3.11}$$

If the pixel $(\overline{x}(y), y)$ satisfies the above constraint then $(X,Y)_j$ is updated by adding the point $(\overline{x}(y), y)$, and the average head distance $\overline{d}_j$ is recalculated.

A vertical head axis H_i is classified as a detected head if it is closer than 5 meters, is between $20\,cm$ and $30\,cm$ long, and is rotated by less than $35°$ from Oy axis in the Cartesian coordinate system. These constraints can be expressed using simple geometry in the manner similar to the above.

A few examples of multiple head detection using this method are shown in figure 3.49.

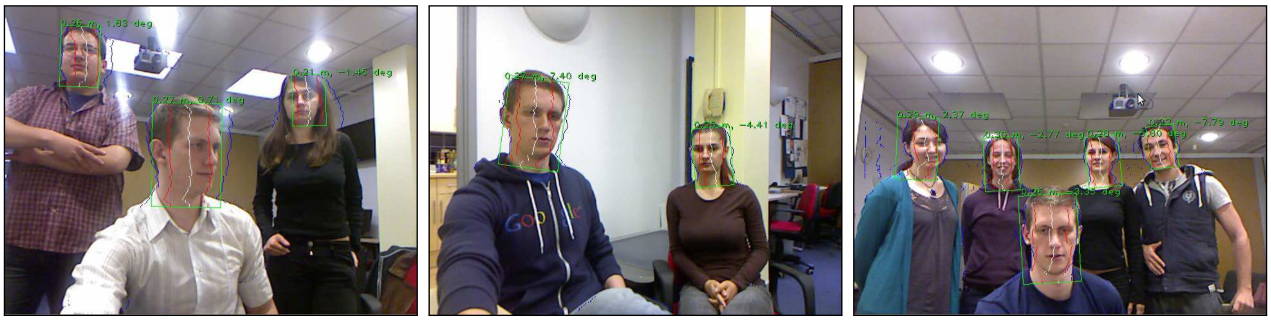


Figure 3.49: Multiple head detection examples using the proposed extension of Garstka and Peters' method.

3.3.1.3 RGB data based improvements to human subject detection

The head detection technique introduced above is very simple, and while it does not produce many false negatives (under the upright human orientation assumption), it often classifies other objects as human heads. In order to reduce the false positive count without too severely compromising the detection rate, a technique with orthogonal failure modes is required.

The most natural way of detecting human subjects from the RGB data would be to perform human/face detection. However, humans might not be fully visible in Kinect's field of view, or might not be facing the camera. Since the event photographer robot should be capable of taking both frontal and profile face pictures, multiple face detector cascades would likely have to be used, using a significant amount of computational complexity budget for a mobile robot.

To keep the complexity low and still reduce the false positive rate of the extended head detection method, skin detection in RGB image is used. More specifically, after detecting a candidate head region using the method described above, the amount of skin-colour pixels in the corresponding RGB image region is examined. If the area occupied by the skin is under a pre-set threshold, the candidate head is rejected.

To perform this task, two skin detection methods are explored: a Bayesian classifier trained off-line on a very large scale skin/non-skin image dataset, and an on-line skin detector trained using skin histograms obtained from a small set of face detections using Viola and Jones (2001) detector. Both of these methods are briefly explained below.

Passive skin-detection using a Bayesian classifier As described by Fleck et al. (1996), the human skin colour is tightly clustered in the colour space, since the human skin hues have a very restricted range (the hue is mostly induced by yellow/brown colour melanin and red colour hemoglobin in blood), and is not very strongly saturated. This indicates that even simple classifiers can achieve good performance. This is verified empirically in the analysis performed by Jones and Rehg (2002), where they have discovered that a histogram-based Bayesian classifier outperforms a more sophisticated Gaussian Mixtures Model (GMM) when trained on a very large scale dataset (containing nearly a billion pixels, each hand-labelled as skin or non-skin).

Because of these reasons, a histogram-based Bayesian classifier similar to Jones and Rehg's is implemented and trained for the use in a robot photographer. Since the skin colour classification performance is largely independent of the colour space used as described by Phung et al. (2005)[10], RGB histograms are used for the classifier's training since the colour input from the Kinect sensor (used for the robot) is provided in the RGB format, thereby avoiding unnecessary format conversions.

Given an input pixel with specific $R = r$, $G = g$ and $B = b$ values (in the equations below shortened as *rgb*), the Bayesian classifier models the probabilities $\Pr(skin|rgb)$ and $\Pr(\neg skin|rgb)$ as

$$\Pr(skin|rgb) = \frac{\Pr(rgb|skin)\Pr(skin)}{\Pr(rgb)},$$
$$\Pr(\neg skin|rgb) = \frac{\Pr(rgb|\neg skin)\Pr(\neg skin)}{\Pr(rgb)}.$$

which allows the posterior ratio to be expressed as

$$\underbrace{\frac{\Pr(skin|rgb)}{\Pr(\neg skin|rgb)}}_{\text{posterior ratio}} = \underbrace{\frac{\Pr(rgb|skin)}{\Pr(rgb|\neg skin)}}_{\text{likelihood ratio}}\underbrace{\frac{\Pr(skin)}{\Pr(\neg skin)}}_{\text{prior ratio}}. \tag{3.12}$$

Then, the *likelihood* ratio can be used to classify a given *rgb* pixel in a probabilistically-sound framework (*i.e.* based on $\Pr(skin|rgb) \geq_? \Pr(\neg skin|rgb)$) using the following observation:

$$\Pr(skin|rgb) \geq \Pr(\neg skin|rgb) \Leftrightarrow$$
$$\frac{\Pr(skin|rgb)}{\Pr(\neg skin|rgb)} \geq 1 \Leftrightarrow$$
$$\frac{\Pr(rgb|skin)\Pr(skin)}{\Pr(rgb|\neg skin)\Pr(\neg skin)} \geq 1 \Leftrightarrow$$
$$\frac{\Pr(rgb|skin)}{\Pr(rgb|\neg skin)} \geq \theta,$$

where $\theta = \frac{\Pr(\neg skin)}{\Pr(skin)}$. Another way of interpreting θ is as a threshold that controls the trade-off between detection/misclassification rates which can either be set empirically or learned from a held-out set.

The maximum likelihood (MLE) estimates for the class-conditional likelihood probabilities $\Pr(rgb|skin)$ and $\Pr(rgb|\neg skin)$ can be obtained from a supervised training set by creating skin and non-skin histograms from all tagged RGB pixels (H_s and H_n respectively) and calculating

$$\Pr(rgb|skin) = \frac{H_s[r,g,b]}{|H_s|} \qquad \Pr(rgb|\neg skin) = \frac{H_n[r,g,b]}{|H_n|},$$

where $H[r,g,b]$ is the pixel count in bin $R = r$, $G = g$, $B = b$ of the histogram H, and $|H|$ is the total pixel count in histogram H.

An example colour input processed using this approach is shown in figure 3.50.

[10]Phung et al. tested a histogram-based Bayesian classifier (similar to the one described in this section) using histograms obtained from RGB, HSV, YCbCr, and CIE-Lab colour spaces, and found that the performance of all classifiers was almost the same in all colour spaces.

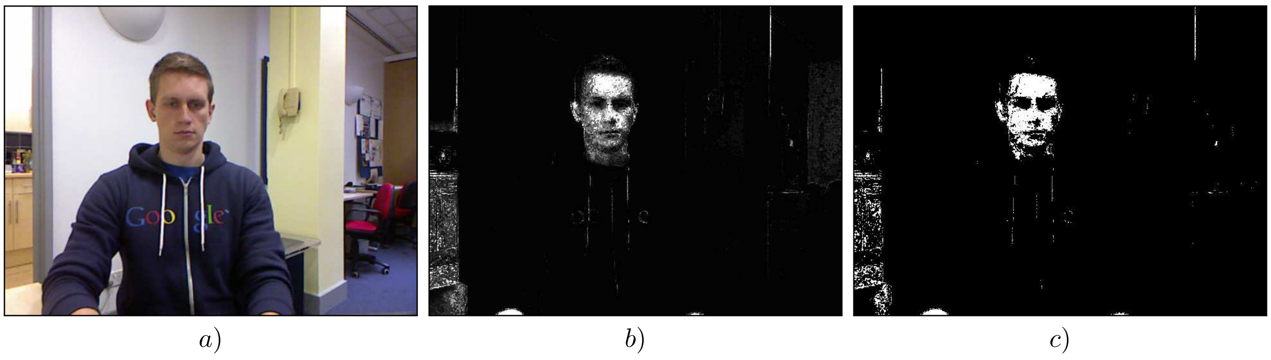


Figure 3.50: Sample output produced by the histogram-based Bayesian skin classifier (Jones and Rehg, 2002). Image $a)$ shows the original input, image $b)$ shows $\frac{\Pr(skin|rgb)}{\Pr(\neg skin|rgb)}$ where pixel's probability corresponds to its intensity, and image c shows the binary skin map obtained after thresholding this probability ratio.

Active skin-detection using a kernel logistic regression classifier with Viola-Jones face detector Another proposed approach for skin detection by an event photographer robot involves building a new skin model for each of the new environments in which the robot is placed.

To achieve this, n faces are detected over a sequence of frames using the frontal and profile face detectors by Viola and Jones (2001), described in section 3.1.1. For each of the detected face rectangles, a binary mask is applied to segment the image into face oval/background regions, and the pixel hue histograms are assembled in each of the regions, as illustrated in figure 3.51. Then these histograms are used as feature vectors in kernel logistic regression (KLR) classifier training.

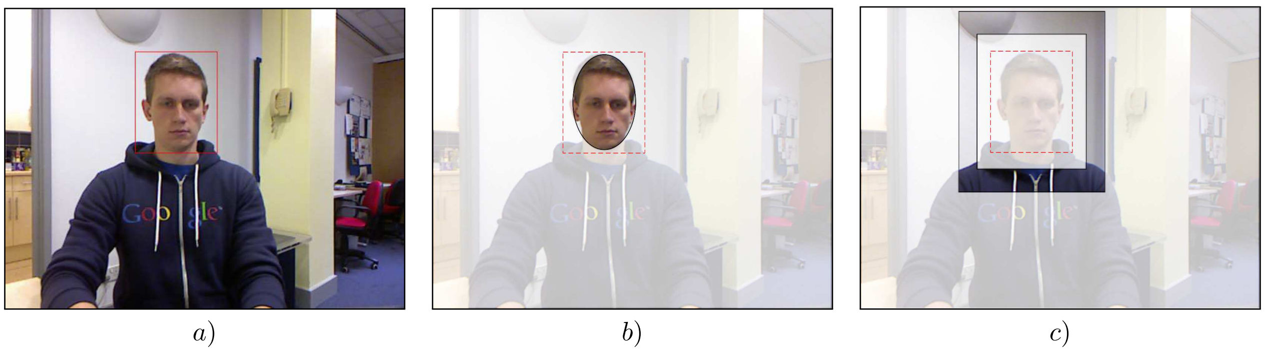


Figure 3.51: Feature extraction process for the skin kernel logistic regression classifier. Image $a)$ shows the original RGB input, together with the face rectangle detected using Viola and Jones's (2001) method. Image $b)$ shows the skin region obtained by rescaling the face rectangle to 95% of its size, setting the width/height ratio to 2:3 and fitting an ellipse to the resulting rectangle. Image $c)$ shows the background region obtained by rescaling the face rectangle to 178% of its size and subtracting from it a face rectangle expanded to 133% of the initial size.

After the training, the depth-based head detections can be verified by applying the same oval binary mask to the detected head rectangle, constructing a hue histogram $\boldsymbol{h}$ from the face region and applying the KLR classifier to obtain $\Pr(skin|\boldsymbol{h})$. The overall detection is then classified as a head if an only if $\Pr(skin|\boldsymbol{h}) > \theta$, where θ is a user-specified threshold.

Given a normalized hue histogram $\boldsymbol{h}$, the logistic regression classifier with the kernel $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$ models the probability that this histogram belongs to a skin class as $\Pr(skin|\boldsymbol{h}) = \sigma(\boldsymbol{w}^T\phi(\boldsymbol{h}))$, where $\boldsymbol{w}$ is the weight vector, σ is a logistic sigmoid function and ϕ is a map from the input space to the feature space. The complementary probability $\Pr(\neg skin|\boldsymbol{h})$ can be obtained by calculating $\Pr(\neg skin|\boldsymbol{h}) = 1 - \Pr(skin|\boldsymbol{h}) = \sigma(-\boldsymbol{w}^T\phi(\boldsymbol{h}))$.

Having obtained n training examples $\mathcal{D} = \{(\boldsymbol{h}_1, t_1), ..., (\boldsymbol{h}_n, t_n)\}$ (where $t_i = 1$ if the histogram is of the skin region, and $t_i = 0$ otherwise), the *maximum-a-posteriori* (MAP) estimate of the weight vector $\boldsymbol{w}$ can be obtained in the following way:

$$\begin{aligned}\boldsymbol{w}_{\text{MAP}} &= \arg\max_{\boldsymbol{w}} \Pr(\boldsymbol{w}|\mathcal{D}) \\ &= \arg\max_{\boldsymbol{w}} \Pr(\mathcal{D}|\boldsymbol{w}) \Pr(\boldsymbol{w}).\end{aligned}$$

Let $\mathcal{L}(\boldsymbol{w}) = -\log(\Pr(\mathcal{D}|\boldsymbol{w}) \Pr(\boldsymbol{w}))$, then equivalently $\boldsymbol{w}_{\text{MAP}} = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$. Modelling the weight prior $\Pr(\boldsymbol{w})$ by a zero-mean Gaussian with variance λ^2 and assuming the independence between individual training examples allows $\mathcal{L}(\boldsymbol{w})$ to be rewritten as

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) &= -\log\left[\left(\prod_i \Pr(t_i|\boldsymbol{h}_i)\right)\exp\left(\frac{-\boldsymbol{w}^T\boldsymbol{w}}{2\lambda^2}\right)\right] \\ &= -\sum_i \left[t_i \log\sigma(\boldsymbol{w}^T\phi(\boldsymbol{h}_i)) + (1-t_i)\log(1-\sigma(\boldsymbol{w}^T\phi(\boldsymbol{h}_i))\right] + \frac{\boldsymbol{w}^T\boldsymbol{w}}{2\lambda^2}.\end{aligned}$$

By the Representer Theorem (*e.g.* see Schölkopf et al. (2001)), the weight vector can be expressed as a linear combination of training examples projected into the feature space, *i.e.* $\boldsymbol{w} = \sum_i \alpha_i\phi(\boldsymbol{h}_i)$, for some $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$, hence the negative log posterior of the weights $\mathcal{L}(\boldsymbol{w})$ can be rewritten as

$$\mathcal{L}(\boldsymbol{\alpha}) = -\sum_i [t_i \log\sigma(\textstyle\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i)) + (1-t_i)\log(1-\sigma(\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i)))] + \frac{\sum_{i,j}\alpha_i\alpha_j K(\boldsymbol{h}_i,\boldsymbol{h}_j)}{2\lambda^2}. \tag{3.13}$$

This quantity can be minimized using the resilient backpropagation (iRprop$^-$, Riedmiller (1994)) variant of the gradient descent algorithm, fully described in the implementation section (see listing 4.2.3.1).

3.3.1.4 Computational performance improvements using a modification of the continuously adaptive mean-shift tracker

To further reduce the computational complexity requirements of head/skin detection methods described above, a depth-data based extension of the continuously adaptive mean-shift tracking algorithm (CAMShift, Bradski (1998)) is employed to exploit the spatial locality of humans over a sequence of frames.

The original CAMShift algorithm is largely based on the *mean shift* algorithm by Fukunaga and Hostetler (1975), which provides a non-parametric way to climb the gradient of a given probability distribution to find the nearest mode. In the extension by Bradski, the mode of the distribution and the size of search window are re-approximated at each input frame, using the zeroth and first spatial horizontal/vertical moments of the probability distribution. This allows the probability distribution of the tracked object's location to be recomputed for each frame.

As shown in line 12 in algorithm 3.3.1.1, the new search window location is obtained by calculating the center of probability mass under the old search window at each iteration of the algorithm. Similarly, the size of the new search window (line 14) is obtained by observing that the zeroth moment approximates the area of the distribution under the search window, hence by assuming a rectangular search window, a square root of the zeroth moment approximates the length of a side of the window. The multiplicative constant (obtained empirically) ensures the

Algorithm 3.3.1.1 A variant of the continuously-adaptive mean shift algorithm. Given the initial location of the search window (x, y), the initial window's size $w \times h$ and the convergence threshold θ, it positions the search window at the nearest dominant mode of the probability distribution P.

```
CONTINUOUSLY-ADAPTIVE-MEAN-SHIFT($P, (x, y), w \times h, \theta$)
1  // Initialize the search window center
2  $(x'_c, y'_c) \leftarrow (x, y)$

3  repeat
4      for one or more iterations
5          $(x_c, y_c) \leftarrow (x'_c, y'_c)$

6          // Find the 0th moment of $P$ under the search window
7          $M_{00} \leftarrow \sum_{|x| \le w/2, |y| \le h/2} P(x_c + x, y_c + y)$
8          // Find the 1st horizontal and vertical spatial moments
9          $M_{10} \leftarrow \sum_{|x| \le w/2, |y| \le h/2} xP(x_c + x, y_c + y)$
10         $M_{01} \leftarrow \sum_{|x| \le w/2, |y| \le h/2} yP(x_c + x, y_c + y)$

11         // Update the object's center position
12         $(x'_c, y'_c) \leftarrow \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right)$

13     // Update the search window size
14     $w \leftarrow 2\sqrt{M_{0,0}}$
15     $h \leftarrow \frac{3}{2}w$
16 until JACCARD-COEFFICIENT(old search window, new search window) $< \theta$
```

sufficient expansion of the search window for CAMShift algorithm to be able to track the whole object, instead of "locking" onto the disconnected parts of the probability distribution.

Finally, the Jaccard coefficient is used as a method's convergence condition. Jaccard's coefficient measures the overlap between two rectangles A and B, and is defined as the ratio between the areas of their intersection and union, *i.e.* JACCARD-COEFFICIENT$(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

While the original CAMShift algorithm by Bradski (1998) uses the probability distribution obtained from the colour hue distribution in the image, in this project it is adapted to use the depth information. In particular, the constraints in equation (3.8) that Garstka and Peters (2011) use to reject such local horizontal minima which could not possibly lie on the vertical head axis, are used to define the following degenerate head probability:

$$\Pr(\textit{head}\,|(x, y)) = \begin{cases} 1, & \text{if constraints in equation (3.8) are satisfied for pixel } (x, y), \\ 0, & \text{otherwise.} \end{cases} \tag{3.14}$$

An example of this method in action is shown in figure 3.52, while the interaction between the detection/tracking components is given in detail in section 4.2.3.2.

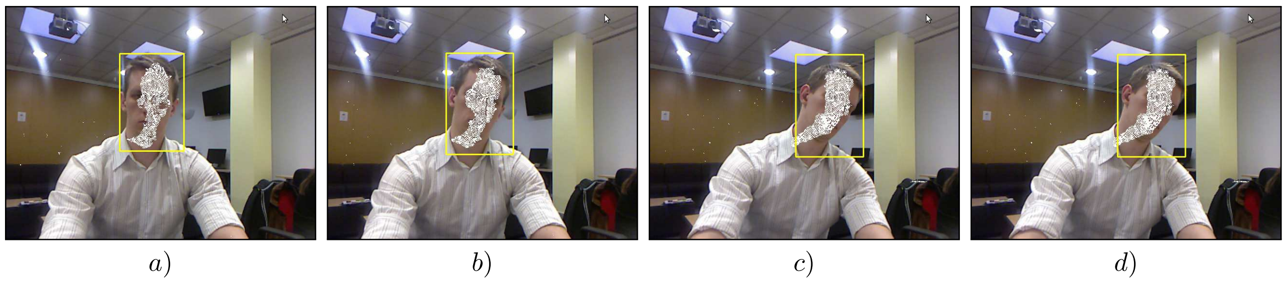


Figure 3.52: Head tracking with extended CAMShift algorithm using depth information. Images a)–d) show the tracked head rectangle (coloured in yellow) overlaid over the input RGB images in a sequence of frames. Pixels with non-zero probability in the head distribution from equation (3.14) are rendered in white.

3.3.2 Proposed obstacle avoidance method

The proposed obstacle avoidance approach for the event robot photographer is based on the work done by Boucher (2012). It was chosen due to its suitability for the random navigation mode (which the event photographer uses to wander around in the environment) and its computational efficiency.

This approach consists of three main steps: i) the prior knowledge about the sensor's position and pitch angle is used to translate and rotate the point cloud *s.t.* the floor plane is described by the equation $z \approx 0$ (under the flat floor assumption), ii) the region of interest in front of the robot is cropped out from the transformed point cloud, and iii) the robot is navigated away from the centroid of remaining points, if there are any. These steps, together with the proposed extensions, are detailed below.

Let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1,\ldots,n}$ be the depth point cloud obtained from the RGB-D sensor, where each $\boldsymbol{x}_i = [x_i, y_i, z_i]^T$.

As a preprocessing step, this depth point cloud is subsampled using a voxel grid filter, to reduce the sensor's noise, and therefore the computational complexity of the subsequent steps, in the following way. Let δ be the length of a single cell's side in the subsampling grid. Then the subsampled point cloud $\overline{\mathcal{D}}$ is composed of the centroids of the points in δ-sized cells, *i.e.*

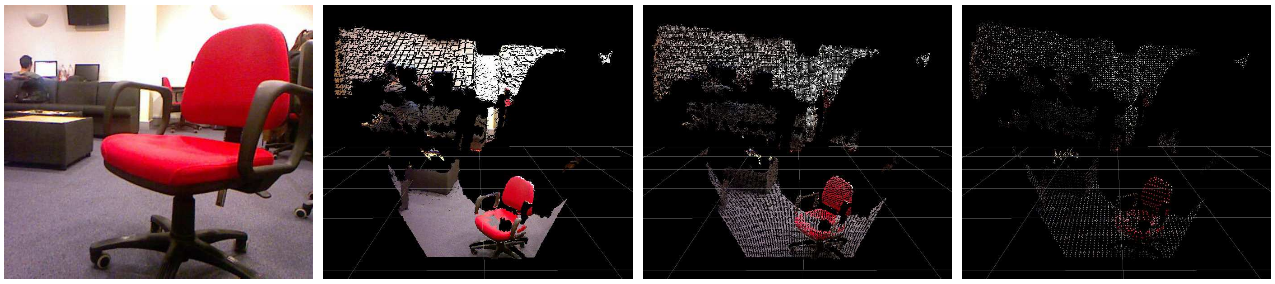
$$\overline{\mathcal{D}} = \left\{ \frac{1}{|C_{x,y,z}|} \sum_{\boldsymbol{x} \in C_{x,y,z}} \boldsymbol{x} \,\middle|\, x \in \left\{1, \ldots, \frac{W}{\delta}\right\}, y \in \left\{1, \ldots, \frac{H}{\delta}\right\}, z \in \left\{1, \ldots, \frac{D}{\delta}\right\} \right\},$$

where $W \times H \times D$ is the size of the input point cloud, and $C_{x,y,z}$ is the set of points in a cell (x, y, z), *i.e.* $C_{x,y,z} = \left\{\boldsymbol{x}_i \in \mathcal{D} \,\middle|\, |x_i - x| < \frac{\delta}{2}, |y_i - y| < \frac{\delta}{2}, |z_i - z| < \frac{\delta}{2}\right\}$.

The input point cloud before and after the voxel grid filtering with different cell sizes δ is shown in figure 3.53.

After filtering the point cloud, the subsampled cloud $\overline{\mathcal{D}}$ is rotated and translated using the knowledge about the RGB-D sensor's position and tilt angle *s.t.* the floor plane is described by equation $z \approx 0$. In Boucher's approach the Kinect sensor is aligned parallel to the ground plane, hence only translation is necessary.

In the modified approach, the accelerometer embedded into Kinect sensor is used to measure the angle θ between the ground plane and the optical axis of the Kinect sensor at each input depth frame. The translation vector $\boldsymbol{t} = [t_x, t_y, t_z]^T$ that describes the sensor's position *w.r.t.* robot's base is measured empirically. Define the rotation



$a)$ RGB input | $b)$ Point cloud $(307,200$ points$)$ | $c)$ $\delta = 2\,cm$ $(36,491$ points$)$ | $d)$ $\delta = 5\,cm$ $(9,860$ points$)$

Figure 3.53: Point cloud processing steps in the proposed obstacle detection method. Image $a)$ shows the original RGB-D point cloud, obtained from the Kinect sensor. Images $b) - d)$ show the same point cloud filtered using the voxel grid filter with varying cell sizes δ.

matrix $\boldsymbol{R}_\theta$ around x-axis (in Kinect's coordinate system) as

$$\boldsymbol{R}_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & sin(\theta) & \cos(\theta) \end{bmatrix}$$

Consider the point representation in homogeneous coordinates (*i.e.* let every point $\boldsymbol{x}_i = [x_i/w_i, y_i/w_i, z_i/w_i]^T$ in $\overline{\mathcal{D}}$ be represented as $\boldsymbol{x}'_i = [x_i, y_i, z_i, w_i]^T$). Then the transformation $\boldsymbol{P}$ that aligns the floor plane with the plane $z = 0$ can be represented as

$$\boldsymbol{P} = \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{\boldsymbol{R}_\theta} & \boldsymbol{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right],$$

i.e. the transformed point cloud $\mathcal{D}_t = \{\boldsymbol{P}\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \overline{\mathcal{D}}\}$. This transformation is illustrated in part $c)$ of figure 3.54.

The next step in the obstacle detection process described by Boucher is to filter the transformed point cloud $\mathcal{D}_t$ to extract the region of interest immediately in front of the robot. Presence of points in this region would indicate obstacles. This is achieved by using further prior knowledge about the robot's size. Let $w \times h$ be the robot's width and height respectively. Furthermore, let d be the depth of the ROI, which determines how far ahead of the robot the obstacle has to be, for the robot to start avoiding it. Then the region of interest $\mathcal{D}_{ROI}$ can be obtained by calculating

$$\mathcal{D}_{ROI} = \left\{\boldsymbol{x}_i \in \mathcal{D}_t \;\middle|\; |x_i| < \frac{w}{2}, 0 < y_i < h, z_i < d\right\}.$$

The resulting point cloud is illustrated in part $d)$ of figure 3.54.

In order to avoid the obstacles in the extracted region of interest, the turn direction for the robot is generated based on the horizontal location of the centroid of $\mathcal{D}_{ROI}$. More specifically, define the centroid of ROI $\overline{\boldsymbol{x}}_{ROI}$ as

$$\overline{\boldsymbol{x}}_{ROI} = \frac{1}{|\mathcal{D}_{ROI}|} \sum_{\boldsymbol{x}_i \in \mathcal{D}_{ROI}} \boldsymbol{x}_i, \text{ if } |\mathcal{D}_{ROI}| > 0.$$

Then the presence of an obstacle is determined by the condition $|\mathcal{D}_{ROI}| > 0$. If this condition is satisfied then the turn direction for the robot is generated by examining the sign of the x_{ROI} coordinate of the centroid $\overline{\boldsymbol{x}}_{ROI} = [x_{ROI}, y_{ROI}, z_{ROI}]^T$: if $x_{ROI} < 0$ then the obstacle lies on the left side of the robot hence a right turn signal is generated, and *vice versa*.

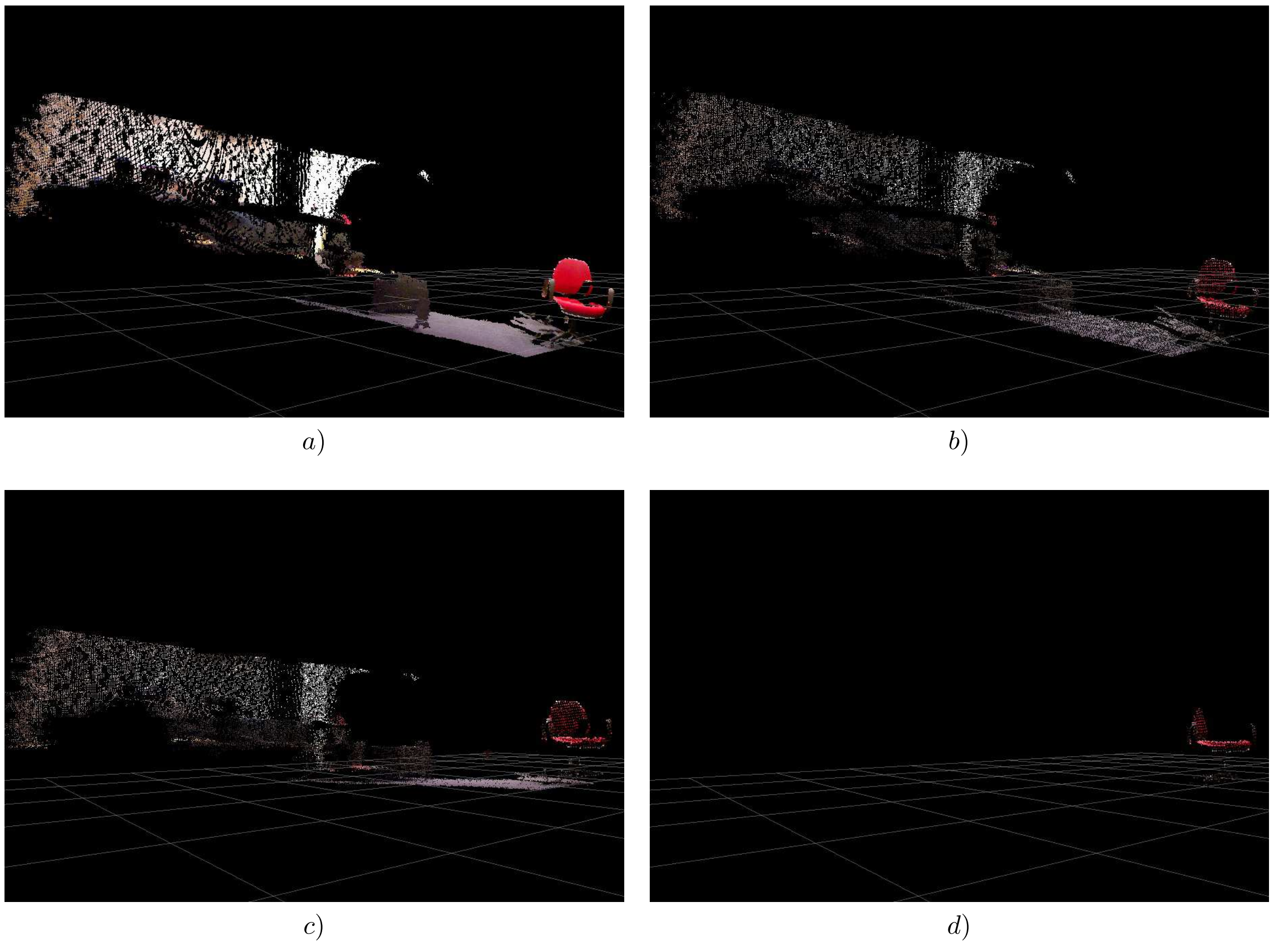


Figure 3.54: Point cloud processing steps in the proposed obstacle detection method. Image $a)$ shows the original RGB-D point cloud, obtained from the Kinect sensor. Image $b)$ shows the same point cloud filtered using the voxel grid filter with $\delta = 3\,cm$. Image $c)$ shows the point cloud transformed by aligning the ground plane with the plane $z = 0$. Image $d)$ shows the obtained region-of-interest point cloud; points in this region are considered as obstacles.

To reduce the impact of sensor noise (arising due to external IR radiation, reflective object surfaces, inaccurate disparity measurements and so on), Boucher proposes the temporal smoothing of the ROI size $|\mathcal{D}_{ROI}|$ by calculating a moving average in the following way.

Let $|\mathcal{D}^t_{ROI}|$ be the size of the ROI at time t. Then the obstacle presence criterion smoothed using a moving-average can be expressed as

$$\frac{1}{n}\sum_{i=0}^{n-1}|\mathcal{D}^{t-i}_{ROI}| > 0.$$

To prevent the robot from getting stuck in an oscillating loop when facing a large obstacle, it is prohibited from changing the direction of the turn once it has started turning, as suggested by Boucher. In other words, after the robot starts to turn to avoid a detected obstacle, it must continue turning in the same direction until the path in front of it clears sufficiently to be able to drive forward.

Also, to avoid turning into another obstacle, which could cause the depth data to disappear making the robot think that there is nothing but empty space ahead (when in reality it is standing right in front of the obstacle) the second improvement proposed by Peasley and Birchfield (2013) is used. Namely, if the unfiltered point cloud $\mathcal{D}$ covers less than $m\%$ of the depth image, then it is assumed that the robot is facing a nearby large obstacle, and a turn direction is issued.

The overall obstacle detection and avoidance process is summarized in an UML activity diagram in figure 3.55, and its source code is provided in appendix A.3.

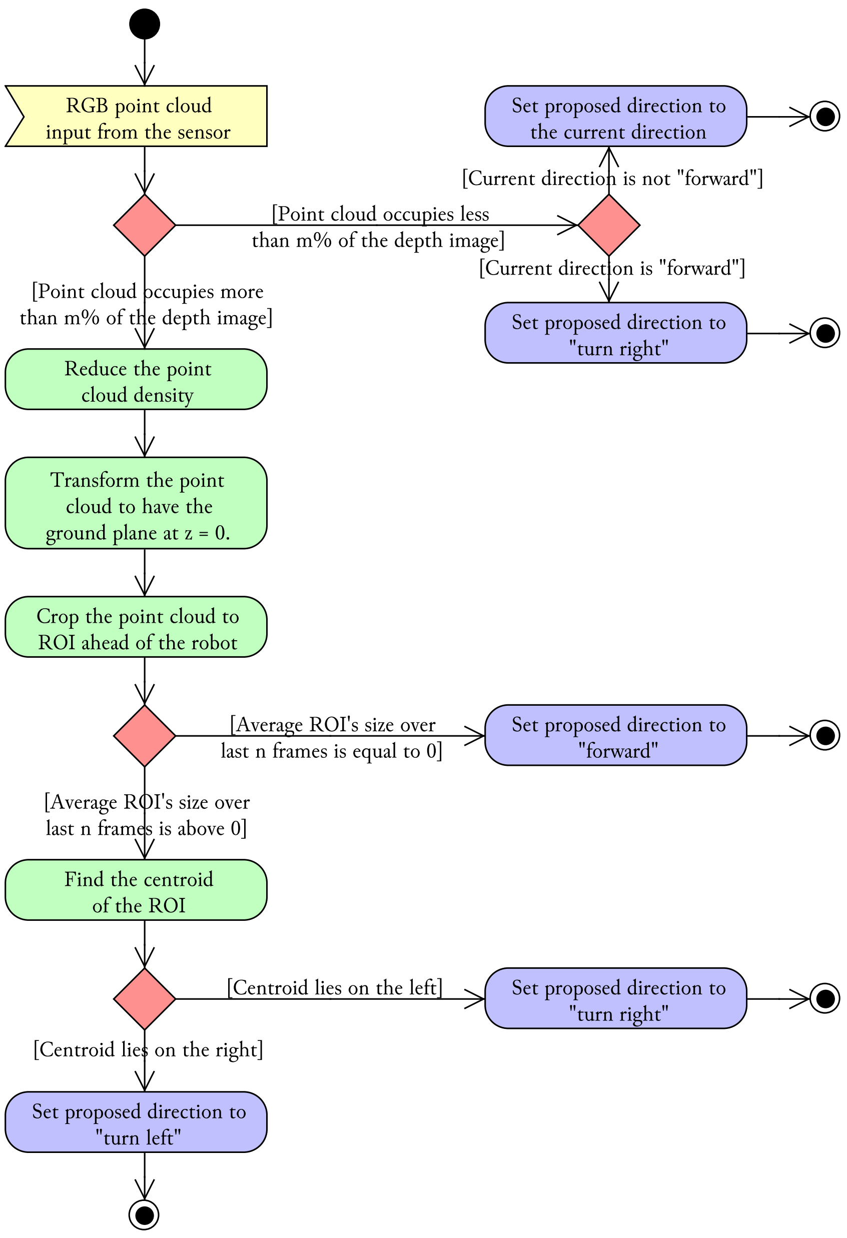

RGB point cloud input from the sensor
[Point cloud occupies less than m% of the depth image]
[Current direction is not "forward"]
Set proposed direction to the current direction
[Current direction is "forward"]
Set proposed direction to "turn right"
[Point cloud occupies more than m% of the depth image]
Reduce the point cloud density
Transform the point cloud to have the ground plane at z = 0.
Crop the point cloud to ROI ahead of the robot
[Average ROI's size over last n frames is equal to 0]
Set proposed direction to "forward"
[Average ROI's size over last n frames is above 0]
Find the centroid of the ROI
[Centroid lies on the left]
Set proposed direction to "turn right"
[Centroid lies on the right]
Set proposed direction to "turn left"

Figure 3.55: A high-level UML activity diagram describing the proposed obstacle detection and avoidance method used by an autonomous event photographer robot.

The following chapter describes the development details of these (and other[11]) methods, including the hardware components and software architecture of the implemented autonomous robot photographer.

[11]Automatic photograph composition and framing, extrinsic and intrinsic calibration of Kinect and photographic cameras, robot's state externalization and so on.

Chapter 4

Development of "Luke": an Event Photographer Robot

This chapter describes a detailed implementation of an autonomous robot photographer, named Luke. First, the hardware components of the developed robot are discussed. Secondly, a high-level software design of the system is presented. Lastly, a detailed description of individual software components is provided.

4.1 Physical structure

Luke (figure 4.1) is built on iClebo Kobuki's base, which is able to carry up to $4\,kg$ payload, has an operating time of around 3 hours and is able to move at the maximum translational and rotational velocities of $65\,cm/s$ and $180\,^{\circ}/s$ respectively. Furthermore, Kobuki's base contains three bumpers (left, center and right) which can be used to provide alternatives to vision-based obstacle avoidance. This base is integrated into the Turtlebot 2 open robotics platform using a kit containing laser-cut mounting plates and aluminium standoffs.

For its vision, Luke uses a Microsoft Kinect RGB-D sensor, which provides both the depth and colour inputs. In order to obtain the depth measures in the scene, Kinect projects a structured light pattern using an embedded IR projector, and captures this pattern using a monochrome CMOS sensor. The distances to objects in the scene are calculated using the triangulation of the IR pattern displacements. The depth-sensing video stream provides VGA resolution (640×480 pixel size) images at around 30 frames per second rate. Each depth reading is represented as a 10-bit value (with a total of 794 possible values), with a hyperbolic relation between these values and the metric depths in the scene. For colour input Kinect uses a RGB camera, which is similarly capable of providing VGA resolution colour images at 30 FPS rate. The sensor has a combined 57° horizontal and 43° vertical field-of-view.

Kinect is attached to the Turtlebot's base at a 10° angle, to be able to track upright standing humans at $1.5\,m$–$2.0\,m$ distance. Since this limits low obstacle detection abilities, the linear velocity of the robot is limited to 10 *m/s* and the bumpers on Kobuki's base are used to provide graceful recovery in the case of collision with a low-lying obstacle.

To take the photographic pictures Luke uses a simple point-and-shoot Nikon COOLPIX S3100 camera, which has a maximum resolution of 14 megapixels, a built-in flash, and supports automatic exposure/ISO sensitivity/white balance settings. This camera is mounted on a lightweight, aluminium König KN-TRIPOD21 tripod (weighing 645 grams), which is attached to the top mounting plate of the robot. The overall size of the robot is approximately $34\,cm \times 135\,cm \times 35\,cm$ ($W \times H \times D$).

For Luke's state externalization, a HTC HD7 smartphone with a 4.3 inch LCD display is mounted onto the robot. The display has a resolution of 480×800 pixels, and is capable of reproducing 24-bit colours. It is used to display Luke's state messages and to show the QR (Quick Response) codes containing the URLs of the pictures that Luke takes and uploads to Flickr. The smartphone also serves as a wireless hotspot, providing a wireless network connection between Luke's on-board computer and a monitoring/debugging station. Furthermore, it provides the internet connection to the on-board computer (for photo uploading to Flickr) by tethering the phone's 3G/EDGE connection over Wi-Fi.

The on-board ASUS Eee PC 1025C netbook serves as Luke's "brain". It has an Intel Atom N2800 1.6 GHz CPU and 1 GB RAM, provides the battery life of around 3 hours and weighs just under $1.25\,kg$. It is running the

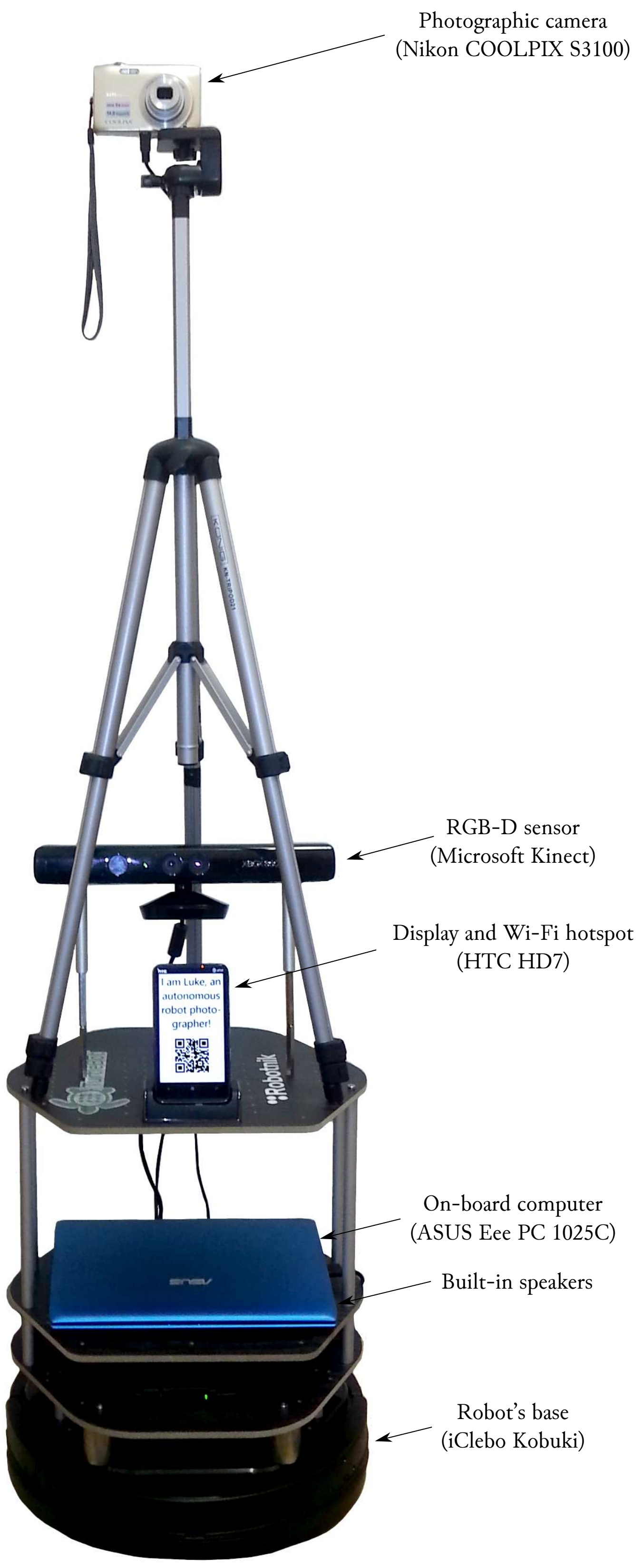


Figure 4.1: Luke, an autonomous event photographer robot.

Groovy Galapagos version of the Robot Operating System framework (ROS, Quigley et al. (2009)) on a Ubuntu 12.04 LTS operating system. All processing (including obstacle avoidance, human subject detection, photographic composition evaluation and so on) is done on this machine.

4.1.1 Power sources

The robot contains two major power sources: a 2,200 *mAh* lithium-ion battery which is enclosed in the Kobuki's base, and a 5,200 *mAh* lithium-ion battery installed in the on-board netbook. The table below summarizes how these sources are used to power individual hardware components of the robot.

Hardware component	Power source
On-board netbook	Netbook's 5,200 *mAh* Li-ion battery
Wheel motors	Base's 2,200 *mAh* Li-ion battery
Phone (display and Wi-Fi hotspot)	Netbook
Photographic camera	Netbook
Kinect RGB-D sensor	Base and netbook

Table 4.1: Energy sources which are used to power individual hardware components of the robot.

During the empirical tests of fully-powered robot, the average discharge times for the netbook's/Kobuki base's batteries were 3 hours and 6 minutes/3 hours and 20 minutes respectively.

Having introduced the main hardware components of Luke and their basic properties, the sections below discuss the software components that provide all of Luke's functionality, starting with a high-level overview of the Robot Operating System.

4.2 Software architecture

All software components of the robot (except the display) are implemented within ROS, an open-source robot operating system. For this reason, the main concepts of ROS are briefly summarized below.

4.2.1 High-level overview of the Robot Operating System

ROS is based on a graph-like architecture: the computation happens in separate processes, called *nodes*, which run in parallel and can be distributed across multiple hosts. A ROS *master* provides naming and registration services which allow the nodes to find each other based on their names during their run-time.

Nodes are connected in a peer-to-peer topology through XML-RPC[1], using programming-language-neutral interface definitions written in interface definition language (IDL). IDL descriptions are strongly typed and can be composed from primitive types, other IDL descriptions (nested arbitrarily deep), or the arbitrary length arrays of these types. The language-neutrality allows individual nodes to be implemented in different programming languages, *e.g.* Luke uses nodes written in both in C++ (majority) and in Python.

Nodes communicate via one-way *message* passing. Two modes of communication are supported by ROS: synchronous and asynchronous. In the synchronous communication mode, a *service* node exposes *request* and *response*

[1]Remote procedure call protocol that uses XML for call encoding and HTTP for call transport.

IDL definitions, together with the service's name. A client node that wants to use this service sends a request message via the ROS master, and blocks until a response message is received.

In the asynchronous communication mode, nodes communicate using the publisher/subscriber pattern. A publisher node provides the IDL description of the messages that it asynchronously broadcasts on a given *topic* name. Multiple subscriber nodes can sign up for the messages on a specific topic. In general, a single node might mix-and-match these communication patterns, *e.g.* it might act as a service provider for a certain type of data, while subscribing to a number of topics and asynchronously publishing other data.

Cohesive sets of nodes (parts of the overall robot's communication and processing graph) can be grouped together into a single ROS *package*, which performs a specific task (*e.g.* provides robot's locomotion). These packages can include the source code of the nodes, IDL message descriptions for topics and services, supporting libraries, documentation and other files. They also include launch instructions, which specify how the cluster of nodes in the package can be integrated into the rest of the computation graph, potentially across different machines.

Finally, to simplify the overall system configuration and maintenance, ROS provides services for global parameter setting and node output logging. For system's behaviour tweaking at run-time, ROS *parameter server* can be used. Nodes can access the parameter server through ROS master and can read/write parameter values from/to a global, hierarchical key-value dictionary. Parameters can also be specified via the launch instructions of the packages. Similarly, ROS nodes can use the system-wide logging server which is capable of logging the node output messages at five verbosity levels in a `printf` style syntax, and provides various tools for on-/off-line log filtering and analysis.

In the next section, the ROS graph of Luke's architectural software design is presented, including the individual nodes and the data that flows in the topics/services over which these nodes communicate.

4.2.2 Architectural system design

Luke's software is architected based on Brooks' (1986) hierarchical levels-of-competence approach. Each of the layers in Luke's software hierarchy is based on the behaviours that Luke can perform:

- The base layer allows Luke to aimlessly wander around the environment, while avoiding collisions with obstacles.
- The second layer suppresses the random wandering behaviour at certain time intervals (adhering to what Brooks called a *subsumption* architecture), and enables Luke to compose, take and upload photographs of people around him.
- The final layer enables Luke to externalize his state i) visually, by showing text messages/QR codes on the attached display, and ii) vocally, by reading state messages out loud using text-to-speech software.

This architecture is illustrated in figure 4.2, and the responsibilities of individual nodes in this architecture are summarized in table 4.2.

4.2.3 Individual node implementations

Individual nodes in the architecture were developed using a separate workstation, with Intel Core i5 CPU, containing two hyperthreaded cores running at 2.3 GHz, and 8 GB of RAM. The workstation was set up to have an identical software configuration as the robot's on-board computer (*i.e.* ROS Groovy Galapagos running on Ubuntu 12.04 LTS).

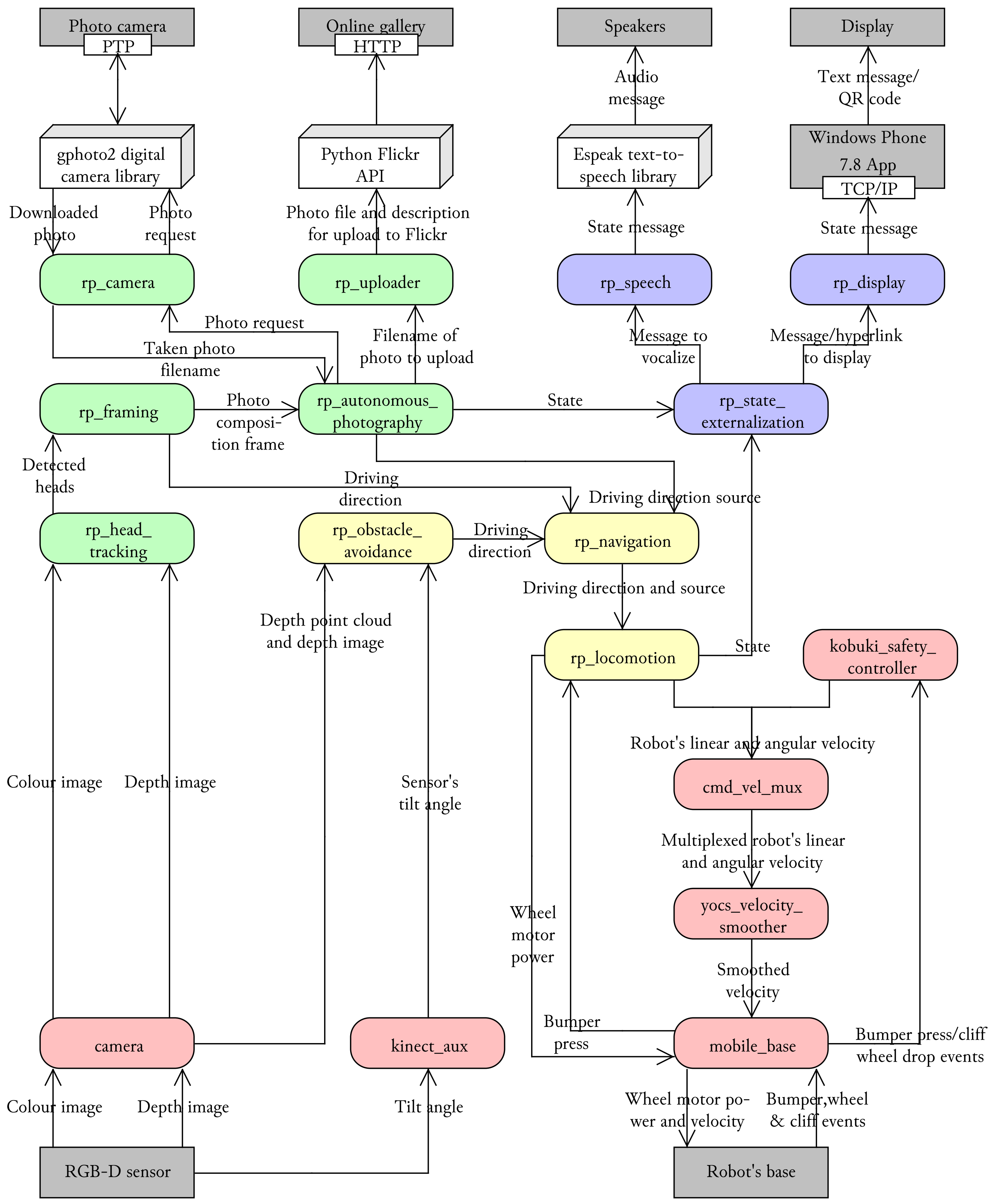


Figure 4.2: A simplified Luke's architectural design diagram, showing ROS nodes (red, green, purple and yellow) together with I/O devices (gray rectangles), and the data that is being passed between them (text on the arrows). All nodes with prefixes "*rp_*" (green, purple and yellow) are the results of the work presented in this thesis, the red nodes are parts of Kobuki/ROS/GFreenect/Kinect AUX libraries. Yellow, green and purple nodes represent the first, second and third Luke's competence levels (corresponding to obstacle avoidance, human tracking/photograph taking and state externalization behaviours). The explanations of individual node functional responsibilities are given in table 4.2.

Table 4.2: Summarized responsibilities of individual nodes comprising Luke's ROS graph.

Node	Main responsibility	Progr. lang.
rp_locomotion	Robot photographer's (RP) locomotion node, which converts driving direction messages and bumper press events into linear/angular velocity messages.	C++
rp_navigation	RP's navigation node, which multiplexes between the competing driving directions proposed by obstacle avoidance and photographic composition (framing) nodes.	C++
rp_obstacle_avoidance	RP's obstacle avoidance node, which uses the point cloud and accelerometer inputs from Kinect to detect obstacles in front of the robot.	C++
rp_head_tracking	RP's head tracking node, which uses the colour and depth inputs from Kinect to detect and track humans in Luke's vicinity.	C++
rp_framing	RP's photographic composition node, which uses the human head locations provided by the head tracking node to calculate the most aesthetically pleasing framing for the picture.	C++
rp_autonomous_photography	Photograph-taking process coordinator node, which issues commands for taking and uploading the pictures.	C++
rp_camera	RP's node which takes pictures using a photographic camera via the gphoto2 library.	C++
rp_uploading	RP's node which uploads taken pictures to the Flickr online gallery, using the Flickr API.	Python
rp_display	RP's display node, which sends status messages/hyperlinks to taken pictures via TCP to a corresponding Windows Phone application that displays them on an attached HTC HD7 phone.	C++
rp_speech	RP's text-to-speech synthesis node, which vocalizes input status messages using the Espeak library.	C++
rp_state_externalization	RP's node responsible for generating vocal/visual status messages about the internal state of the robot.	C++
Robot's state display app for Windows Phone†	This app serves as a counterpart to the *rp_display* node, by rendering the incoming status messages/hyperlinks (the latter ones as QR-codes) on the phone's display.	C#
Camera calibration tool†	This tool is used to calibrate the photographic camera (by removing tangential and radial lens distortion), which is necessary for photographic camera and depth camera alignment.	C++
camera	GFreenect drivers for Kinect sensor, which provide aligned colour and depth images, and point clouds.	-
kinect_aux	Kinect AUX drivers, which provide the Kinect sensor's accelerometer readings (*i.e.* the sensor's tilt angle).	-
mobile_base	iClebo Kobuki base driver, which provides access to the wheel motor power and linear/angular velocities, and bumper press/wheel drop/cliff events.	-
yocs_velocity_smoother	Yujin Robot's Open-Source Control Software (YOCS) velocity and acceleration smoother.	-
cmd_vel_mux	YOCS velocity input multiplexer, which serializes and prioritizes incoming velocity messages (giving preference to the safety controller's messages).	-
kobuki_safety_controller	iClebo Kobuki's safety controller, which keeps track of the base's wheel drop/-cliff/bumper events. In the first case it produces a stop command (zero velocity), in the latter two cases it produces a negative linear velocity.	-

† Not parts of robot photographer's ROS node graph.

For node development and debugging, an Eclipse integrated development environment (IDE) was adapted to work with ROS node build/launch system. For C++ development an Eclipse CDT (C/C++ Development Tooling) extension was set-up, providing visual debugging tools, source navigation, refactoring and other capabilities. To ensure the maintainability of the code, C++ nodes were written adhering to Google's C++ coding standard. An analogous extension for Eclipse called PyDev was set up for node development in Python. Finally, for Windows Phone app development (for the robot's display), the workstation was adapted to dual-boot into Windows 7, where Microsoft Visual Studio 2010 IDE was set up for C# development.

Both IDEs were fully configured to use Git distributed version control and source code management system, with the main repository hosted on a remote server featuring a guaranteed minimum three-copy backup and an immediate hot-swap system in the case of a hardware or software failure.

Having provided the description of the development environment used in this project, the implementation details of individual nodes in each of Luke's competence layers are described below, starting with the random wandering behavioural capability.

4.2.3.1 Random walking with collision avoidance

Luke's capability to randomly wander in the environment without bumping into any static or moving obstacles is implemented in three ROS nodes: *rp_obstacle_detection*, *rp_locomotion* and *rp_navigation*. The implementation of each of these nodes is briefly discussed below.

Obstacle avoidance (rp_obstacle_avoidance) The proposed method to detect and avoid the obstacles (as described in section 3.3.2) is implemented by *rp_obstacle_avoidance* node.

This node starts the processing by obtaining the input point cloud from the *camera/depth_registered/points* topic, provided by the GFreenect library (OpenKinect, 2012) which publishes point clouds at 30 *Hz* frequency. Similarly, the obstacle avoidance node obtains the Kinect's tilt angle from a *kinect_aux/cur_tilt_angle* topic provided by the Kinect AUX library (Dryanovski et al., 2011). This library provides the readings from the Kinect's accelerometer at 20 *Hz* frequency.

Then *rp_obstacle_avoidance* node performs the main point cloud manipulations as described in section 3.3.2 (*viz.* point cloud subsampling, translation/rotation to align the ground plane with the plane $z = 0$, and cropping to the

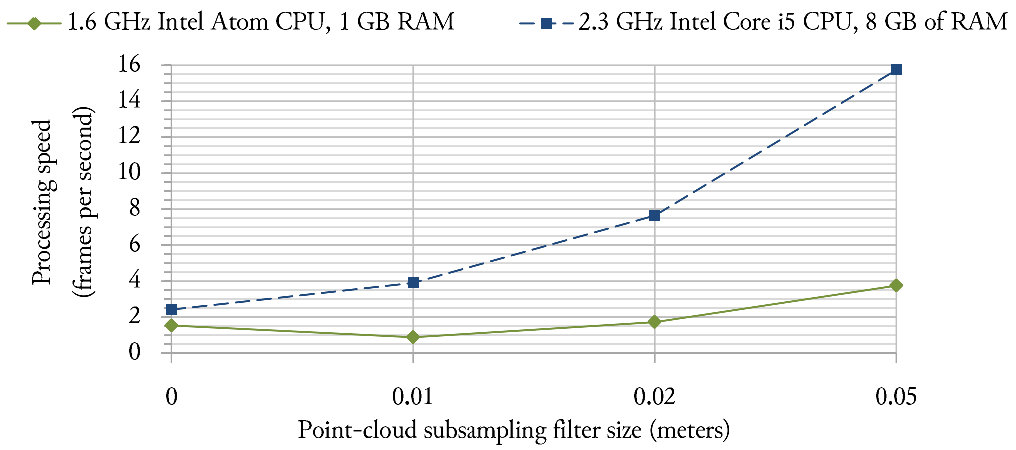


Figure 4.3: Performance evaluation of the obstacle detection and avoidance node (*rp_obstacle_avoidance*) under different point cloud subsampling filter sizes.

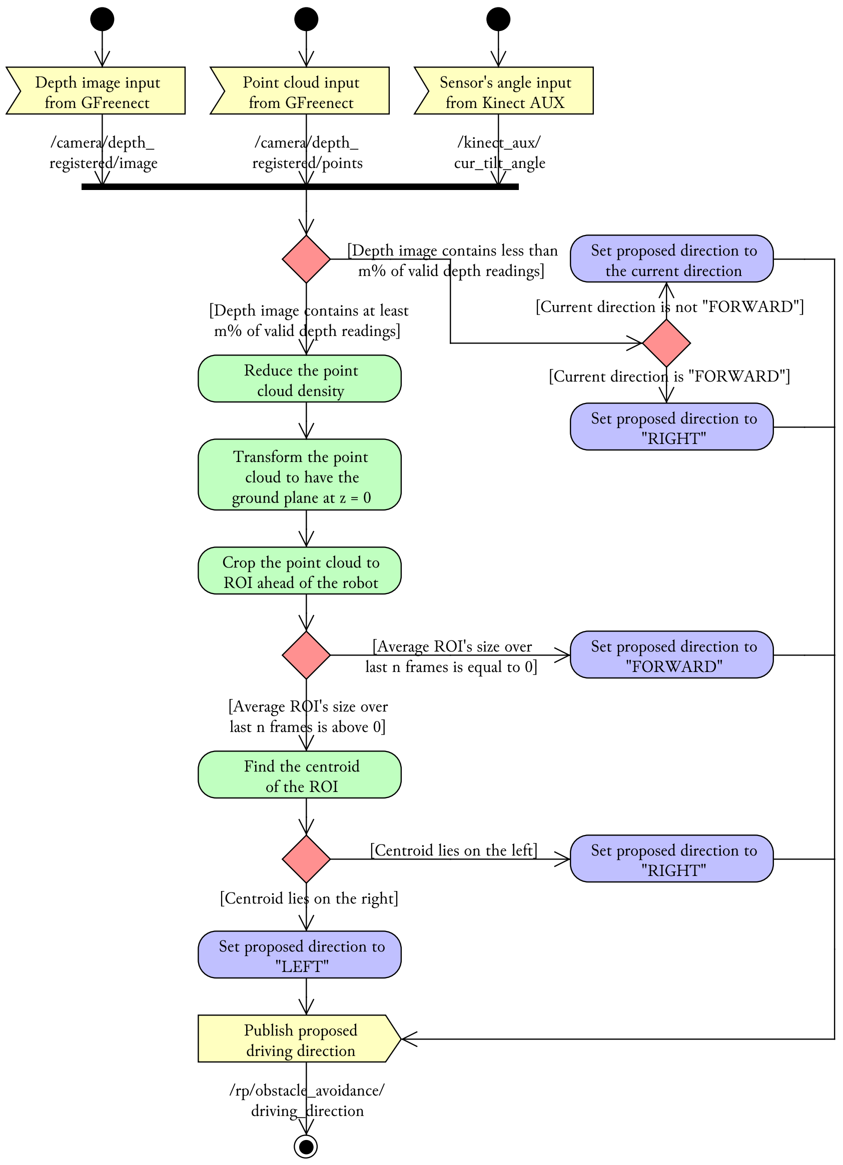

Depth image input from GFreenect
Point cloud input from GFreenect
Sensor's angle input from Kinect AUX
/camera/depth_ registered/image
/camera/depth_ registered/points
/kinect_aux/ cur_tilt_angle
[Depth image contains less than m% of valid depth readings]
Set proposed direction to the current direction
[Current direction is not "FORWARD"]
[Depth image contains at least m% of valid depth readings]
[Current direction is "FORWARD"]
Reduce the point cloud density
Set proposed direction to "RIGHT"
Transform the point cloud to have the ground plane at z = 0
Crop the point cloud to ROI ahead of the robot
[Average ROI's size over last n frames is equal to 0]
Set proposed direction to "FORWARD"
[Average ROI's size over last n frames is above 0]
Find the centroid of the ROI
[Centroid lies on the left]
Set proposed direction to "RIGHT"
[Centroid lies on the right]
Set proposed direction to "LEFT"
Publish proposed driving direction
/rp/obstacle_avoidance/ driving_direction

Figure 4.4: A simplified UML activity diagram of the obstacle avoidance (*rp_obstacle_avoidance*) ROS node.

ROI in front of the robot). All these manipulations are performed using an open-source PCL library (Rusu and Cousins, 2011).

After the point cloud pre-processing, this node calculates the moving average of the ROI in front of the robot over the last n frames. If the average size exceeds zero, then this node generates a turn direction (`LEFT` or `RIGHT`) which steers the robot away from the centroid of points in the ROI. If the ROI over the last n frames is empty, then this node generates `FORWARD` driving direction proposal. These direction proposals are published over the *rp/obstacle_avoidance/driving_direction* topic.

The computational performance of this node largely depends on the point cloud size, which in turn depends on the grid size of the voxel subsampling filter, used in the pre-processing step. The impact of this filter's size to the overall node's performance was measured using two different machines: Luke's on-board netbook (with dual-core 1.6 GHz Intel Atom CPU, 1 GB RAM) and the development workstation (with dual-core hyperthreaded 2.3 GHz Intel Core i5 CPU, 8 GB of RAM). The obtained results are presented in figure 4.3.

Navigation (rp_navigation) The navigation node (*rp_navigation*) multiplexes between the driving direction suggestions provided by the obstacle avoidance (*rp_obstacle_avoidance*) and photograph framing (*rp_framing*) nodes, based on the input from the autonomous photography process control node (*rp_autonomous_photography*).

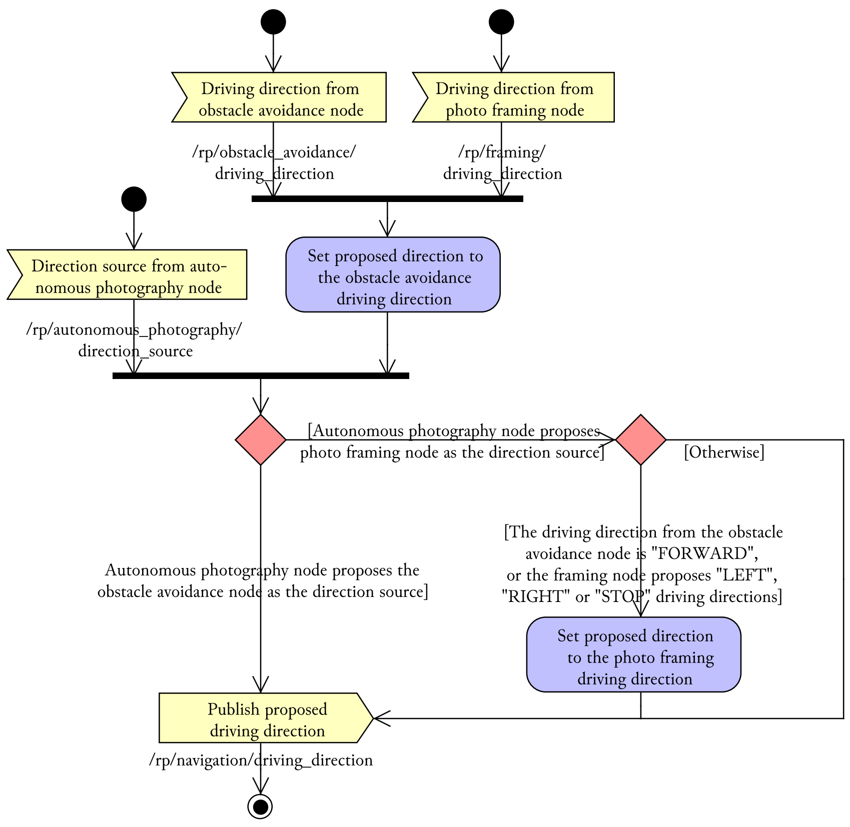

Driving direction from obstacle avoidance node
Driving direction from photo framing node
/rp/obstacle_avoidance/driving_direction
/rp/framing/driving_direction
Direction source from autonomous photography node
Set proposed direction to the obstacle avoidance driving direction
/rp/autonomous_photography/direction_source
[Autonomous photography node proposes photo framing node as the direction source]
[Otherwise]
[The driving direction from the obstacle avoidance node is "FORWARD", or the framing node proposes "LEFT", "RIGHT" or "STOP" driving directions]
Autonomous photography node proposes the obstacle avoidance node as the direction source]
Set proposed direction to the photo framing driving direction
Publish proposed driving direction
/rp/navigation/driving_direction

Figure 4.5: A simplified UML activity diagram of the navigation (*rp_navigation*) ROS node.

In essence, the navigation node enables the framing node to override the random wandering behaviour provided by the obstacle avoidance node, but only when it is safe to do so. In particular, it allows the framing node to override the driving direction if and only if the obstacle avoidance node proposes the `FORWARD` direction, or if the framing node wants to stop/turn in place.

Its simplified UML activity diagram is shown in figure 4.5.

Locomotion (rp_locomotion) Luke's locomotion node (*rp_locomotion*) is responsible for generating initial linear and angular velocity messages, based on multiplexed driving direction generated by the navigation node and on bumper events generated by the Kobuki's base node (*mobile_base*). Its behaviour is illustrated in figure 4.6.

Basically, the locomotion node takes the input driving direction (`FORWARD`, `BACKWARD`, `LEFT`, `RIGHT` or `STOP`) and produces an appropriate linear/angular velocity message. For example, a message corresponding to the direction `FORWARD` would contain linear and angular velocity vectors $\boldsymbol{l} = [\theta_l, 0, 0]^T$ and $\boldsymbol{a} = [0, 0, 0]^T$ respectively, where θ_l is a desired linear velocity parameter (settable through the launch file or through the parameter server). Similarly, if the driving direction is `LEFT` then linear and angular velocity vectors would contain $\boldsymbol{l} = [0, 0, 0]^T$ and $\boldsymbol{a} = [0, 0, \theta_a]^T$, where θ_a is the desired angular velocity parameter.

However, if a bumper event is registered then the locomotion node generates the message containing $\boldsymbol{l} = \mathbf{0}$ and $\boldsymbol{a} = [0, 0, \pm\theta_a]^T$ (where the direction of turn depends on which bumper registered the event). The locomotion node keeps producing this message until the robot turns $\sim 180°$ before continuing its random wandering.

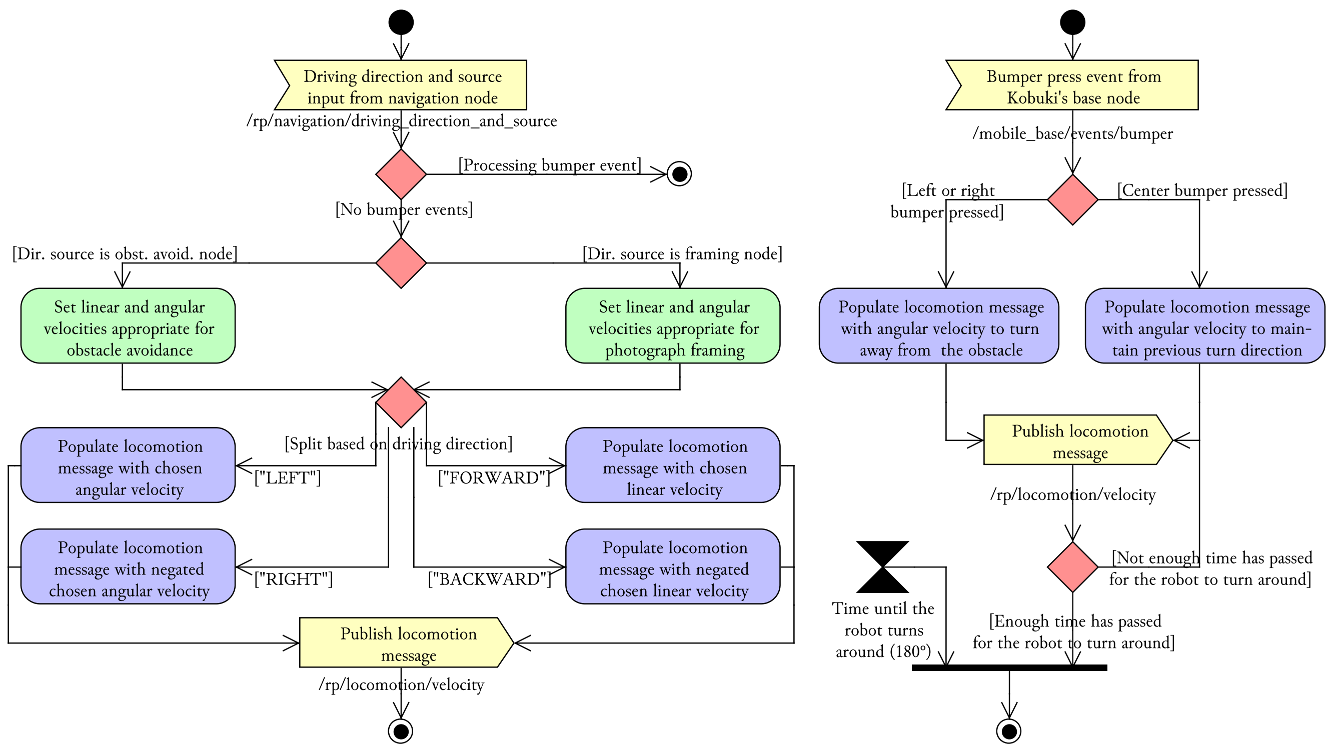

Driving direction and source input from navigation node
/rp/navigation/driving_direction_and_source
[Processing bumper event]
[No bumper events]
[Dir. source is obst. avoid. node]
[Dir. source is framing node]
Set linear and angular velocities appropriate for obstacle avoidance
Set linear and angular velocities appropriate for photograph framing
[Split based on driving direction]
Populate locomotion message with chosen angular velocity
["LEFT"]
["FORWARD"]
Populate locomotion message with chosen linear velocity
Populate locomotion message with negated chosen angular velocity
["RIGHT"]
["BACKWARD"]
Populate locomotion message with negated chosen linear velocity
Publish locomotion message
/rp/locomotion/velocity
Bumper press event from Kobuki's base node
/mobile_base/events/bumper
[Left or right bumper pressed]
[Center bumper pressed]
Populate locomotion message with angular velocity to turn away from the obstacle
Populate locomotion message with angular velocity to maintain previous turn direction
Publish locomotion message
/rp/locomotion/velocity
[Not enough time has passed for the robot to turn around]
Time until the robot turns around (180°)
[Enough time has passed for the robot to turn around]

Figure 4.6: A simplified UML activity diagram of the locomotion (*rp_locomotion*) ROS node.

Once the locomotion message is constructed, it is passed to Yujin Robot's Open-Source Control Software (YOCS) velocity multiplexer node (*cmd_vel_mux*). This node also receives the input from YOCS safety controller (*kobuki_safety_controller*), which generates temporary zero/negative linear velocity messages in the respective cases of wheel drop or cliff/bumper events. The YOCS velocity multiplexer mode prioritizes the input from the safety controller node, hence the robot is further prevented from causing damage to itself, or objects in its environment.

The output produced by YOCS velocity multiplexer is sent to the YOCS velocity smoother (*yocs_velocity_smoother*), which applies the given acceleration/deceleration and velocity limits to input velocities, thereby ensuring a fluid motion of the robot. These smoothed linear/angular velocities are then sent directly to the Kobuki's base node (*mobile_base*), which appropriately sets the robot's wheel speeds.

4.2.3.2 Taking well-composed photographs of humans

Luke's second major behavioural competence involves his ability to *i*) track humans in an unstructured environment, *ii*) take well-composed pictures of them, and *iii*) upload these pictures to an on-line picture gallery. This competence layer is implemented by five ROS nodes: *rp_head_tracking*, *rp_framing*, *rp_camera*, *rp_uploading* and *rp_autonomous_photography*, each of which is discussed in more detail below.

Head detection and tracking (rp_head_tracking) The head detection and tracking node (*rp_head_tracking*) is the most sophisticated node in Luke's ROS graph, consisting of multiple loosely-coupled classes with clearly separated responsibilities (shown in UML class diagram in figure 4.7).

This node implements the human subject detection and tracking methods, proposed in section 3.3.1. In particular, it performs multiple human detection from depth data using an extension to Garstka and Peters' (2011) method,

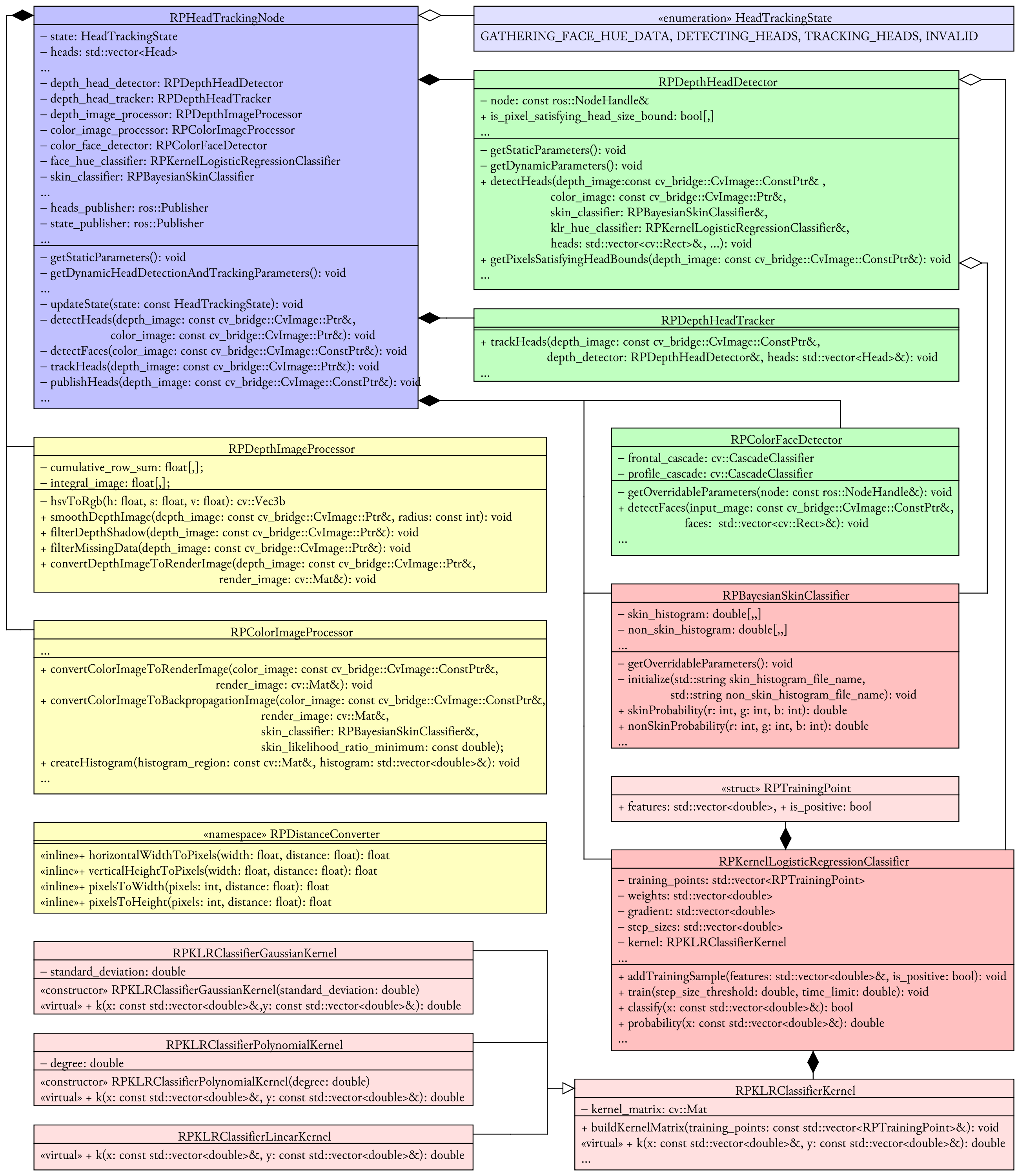

RPHeadTrackingNode
– state: HeadTrackingState
– heads: std::vector<Head>
...
– depth_head_detector: RPDepthHeadDetector
– depth_head_tracker: RPDepthHeadTracker
– depth_image_processor: RPDepthImageProcessor
– color_image_processor: RPColorImageProcessor
– color_face_detector: RPColorFaceDetector
– face_hue_classifier: RPKernelLogisticRegressionClassifier
– skin_classifier: RPBayesianSkinClassifier
...
– heads_publisher: ros::Publisher
– state_publisher: ros::Publisher
...
– getStaticParameters(): void
– getDynamicHeadDetectionAndTrackingParameters(): void
...
– updateState(state: const HeadTrackingState): void
– detectHeads(depth_image: const cv_bridge::CvImage::Ptr&, color_image: const cv_bridge::CvImage::Ptr&): void
– detectFaces(color_image: const cv_bridge::CvImage::ConstPtr&): void
– trackHeads(depth_image: const cv_bridge::CvImage::Ptr&): void
– publishHeads(depth_image: const cv_bridge::CvImage::ConstPtr&): void
...
«enumeration» HeadTrackingState
GATHERING_FACE_HUE_DATA, DETECTING_HEADS, TRACKING_HEADS, INVALID
RPDepthHeadDetector
– node: const ros::NodeHandle&
+ is_pixel_satisfying_head_size_bound: bool[,]
...
– getStaticParameters(): void
– getDynamicParameters(): void
+ detectHeads(depth_image:const cv_bridge::CvImage::ConstPtr& , color_image: const cv_bridge::CvImage::Ptr&, skin_classifier: RPBayesianSkinClassifier&, klr_hue_classifier: RPKernelLogisticRegressionClassifier&, heads: std::vector<cv::Rect>&, ...): void
+ getPixelsSatisfyingHeadBounds(depth_image: const cv_bridge::CvImage::ConstPtr&): void
...
RPDepthHeadTracker
+ trackHeads(depth_image: const cv_bridge::CvImage::ConstPtr&, depth_detector: RPDepthHeadDetector&, heads: std::vector<Head>&): void
...
RPColorFaceDetector
– frontal_cascade: cv::CascadeClassifier
– profile_cascade: cv::CascadeClassifier
– getOverridableParameters(node: const ros::NodeHandle&): void
+ detectFaces(input_mage: const cv_bridge::CvImage::ConstPtr&, faces: std::vector<cv::Rect>&): void
...
RPDepthImageProcessor
– cumulative_row_sum: float[,];
– integral_image: float[,];
– hsvToRgb(h: float, s: float, v: float): cv::Vec3b
+ smoothDepthImage(depth_image: const cv_bridge::CvImage::Ptr&, radius: const int): void
+ filterDepthShadow(depth_image: const cv_bridge::CvImage::Ptr&): void
+ filterMissingData(depth_image: const cv_bridge::CvImage::Ptr&): void
+ convertDepthImageToRenderImage(depth_image: const cv_bridge::CvImage::Ptr&, render_image: cv::Mat&): void
RPBayesianSkinClassifier
– skin_histogram: double[,,]
– non_skin_histogram: double[,,]
...
– getOverridableParameters(): void
– initialize(std::string skin_histogram_file_name, std::string non_skin_histogram_file_name): void
+ skinProbability(r: int, g: int, b: int): double
+ nonSkinProbability(r: int, g: int, b: int): double
...
RPColorImageProcessor
...
+ convertColorImageToRenderImage(color_image: const cv_bridge::CvImage::ConstPtr&, render_image: cv::Mat&): void
+ convertColorImageToBackpropagationImage(color_image: const cv_bridge::CvImage::ConstPtr&, render_image: cv::Mat&, skin_classifier: RPBayesianSkinClassifier&, skin_likelihood_ratio_minimum: const double);
+ createHistogram(histogram_region: const cv::Mat&, histogram: std::vector<double>&): void
...
«struct» RPTrainingPoint
+ features: std::vector<double>, + is_positive: bool
«namespace» RPDistanceConverter
«inline»+ horizontalWidthToPixels(width: float, distance: float): float
«inline»+ verticalHeightToPixels(width: float, distance: float): float
«inline»+ pixelsToWidth(pixels: int, distance: float): float
«inline»+ pixelsToHeight(pixels: int, distance: float): float
RPKernelLogisticRegressionClassifier
– training_points: std::vector<RPTrainingPoint>
– weights: std::vector<double>
– gradient: std::vector<double>
– step_sizes: std::vector<double>
– kernel: RPKLRClassifierKernel
...
+ addTrainingSample(features: std::vector<double>&, is_positive: bool): void
+ train(step_size_threshold: double, time_limit: double): void
+ classify(x: const std::vector<double>&): bool
+ probability(x: const std::vector<double>&): double
...
RPKLRClassifierGaussianKernel
– standard_deviation: double
«constructor» RPKLRClassifierGaussianKernel(standard_deviation: double)
«virtual» + k(x: const std::vector<double>&,y: const std::vector<double>&): double
RPKLRClassifierPolynomialKernel
– degree: double
«constructor» RPKLRClassifierPolynomialKernel(degree: double)
«virtual» + k(x: const std::vector<double>&, y: const std::vector<double>&): double
RPKLRClassifierLinearKernel
«virtual» + k(x: const std::vector<double>&, y: const std::vector<double>&): double
RPKLRClassifierKernel
– kernel_matrix: cv::Mat
+ buildKernelMatrix(training_points: const std::vector<RPTrainingPoint>&): void
«virtual» + k(x: const std::vector<double>&, y: const std::vector<double>&): double
...

Figure 4.7: A simplified UML class diagram of a subset of classes implementing *rp_head_tracker* ROS node. Image pre-processing/utility classes are shown in yellow, depth/colour face detectors and trackers are shown in green, skin classifiers are shown in red, and the node’s “command-and-control” class is shown in purple.

detected head candidate verification using RGB data and Bayesian/kernel logistic regression classifiers, and detected human subject tracking using a modification of continuously-adaptive mean-shift tracker, by Bradski (1998).

The *rp_head_tracking* node persists two main pieces of state between consecutive frames:

- a set of currently detected/tracked heads $\mathcal{H} = \{(\boldsymbol{r}_i, f_i)\}_{i=1,\dots,k}$, where each head i is represented by a rectangle in the image plane $\boldsymbol{r}_i$ and the number of frames f_i since it was last detected, and
- the mode that the *rp_head_tracking* node is operating in.

Depending on whether the Bayesian or kernel logistic regression classifier is used for skin detection, the head detection/tracking node can be in one of $\{$`DETECTING_HEADS`, `TRACKING_HEADS`$\}$ or $\{$`GATHERING_FACE_HUE_DATA`, `DETECTING_HEADS`, `TRACKING_HEADS`$\}$ modes respectively. The behaviour of the *rp_head_tracking* node in each of those modes is further described below, but first the head set $\mathcal{H}$ update mechanism is explained.

Let $\mathcal{H}_t = \{(\boldsymbol{r}_i, h_i)_t\}$ be the set of detected/tracked heads at time t. To obtain the updated set $\mathcal{H}_{t+1}$ (which is initialized to $\mathcal{H}_{t+1} = \mathcal{H}_t$) the following procedure is applied:

- If the node is in `DETECTING_HEADS` mode, then let $\mathcal{R}_{t+1} = \{\boldsymbol{r}_i\}$ be the set of head rectangles detected at time $t+1$. Given this set, the counters of "old" heads which do not have corresponding "new" head detections are incremented:

$$\forall (\boldsymbol{r}_i, h_i)_t \in \mathcal{H}_t. \, (\neg \exists \boldsymbol{r}_j \in \mathcal{R}_{t+1}. \mathit{sufficientlySimilar}(\boldsymbol{r}_i, \boldsymbol{r}_j)) \rightarrow (\boldsymbol{r}_i, h_i)_{t+1} := (\boldsymbol{r}_i, h_i + 1)_t,$$

where $\mathit{sufficientlySimilar}(\boldsymbol{x}, \boldsymbol{y})$ is the predicate that determines whether two rectangles $\boldsymbol{x}$ and $\boldsymbol{y}$ are sufficiently similar. Possible variants of this predicate include the thresholded Jaccard's coefficient ($\mathit{sufficientlySimilar}(\boldsymbol{x}, \boldsymbol{y}) = \frac{|\boldsymbol{x} \cap \boldsymbol{y}|}{|\boldsymbol{x} \cup \boldsymbol{y}|} > \theta$) or a simple rectangle intersection non-emptiness check ($\mathit{sufficientlySimilar}(\boldsymbol{x}, \boldsymbol{y}) = |\boldsymbol{x} \cap \boldsymbol{y}| > 0$, as used in this project).

Then the "old" heads which have corresponding sufficiently similar "new" heads have their histories reset and their head rectangles updated:

$$\forall (\boldsymbol{r}_i, h_i)_t \in \mathcal{H}_t. \, (\exists \boldsymbol{r}_j \in \mathcal{R}_{t+1}. \mathit{sufficientlySimilar}(\boldsymbol{r}_i, \boldsymbol{r}_j)) \rightarrow (\boldsymbol{r}_i, h_i)_{t+1} := (\boldsymbol{r}_j, 0).$$

Since new people might have entered the robot's field-of-view, the "new" heads that do not have corresponding "old" heads are added to the head set:

$$\forall \boldsymbol{r}_i \in \mathcal{R}_{t+1}. \, (\neg \exists (\boldsymbol{r}_j, h_j)_t \in \mathcal{H}_t. \mathit{sufficientlySimilar}(\boldsymbol{r}_i, \boldsymbol{r}_j)) \rightarrow \mathcal{H}_{t+1} := \mathcal{H}_{t+1} \cup \{(\boldsymbol{r}_i, 0)\}.$$

Finally, the heads that were not re-detected for more than n frames are removed from $\mathcal{H}_{t+1}$:

$$\forall (\boldsymbol{r}_i, h_i)_{t+1} \in \mathcal{H}_{t+1}. h_i > n \rightarrow \mathcal{H}_{t+1} := \mathcal{H}_{t+1} \backslash \{(\boldsymbol{r}_i, h_i)_{t+1}\}.$$

In essence this approach is quite similar to the clock-style page replacement algorithms in OS virtual memory management systems, in a sense that the detection algorithm is allowed "second chances". More precisely, a given head must not be found in n subsequent detection attempts before it is removed, thereby reducing the sensitivity of the overall head detection and tracking approach to the sensor's noise and inaccuracies of the head's detection method.

- If the node is in TRACKING_HEADS mode, then the modified Bradski's (1998) CAMShift algorithm is used to track each of the heads individually, as described in section 3.3.1.4.

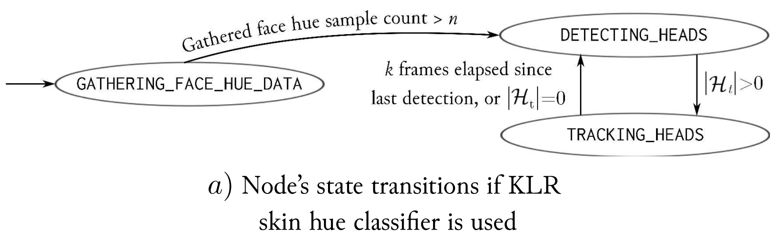
  In particular, for each head $(\boldsymbol{r}_i, h_i)_{t+1} \in \mathcal{H}_{t+1}$ let $\widehat{\boldsymbol{r}}_i$ be the new head rectangle obtained by CAMShift tracker.

  If $|\widehat{\boldsymbol{r}}_i| > 0$ then the head set $\mathcal{H}_{t+1}$ is updated by setting $(\boldsymbol{r}_i, h_i)_{t+1} := (\widehat{\boldsymbol{r}}_i, h_i)_{t+1}$.

  Otherwise, the head is considered lost and is removed from the head set, *i.e.* $\mathcal{H}_{t+1} := \mathcal{H}_{t+1} \backslash \{(\boldsymbol{r}_i, h_i)_{t+1}\}$.

- The head set $\mathcal{H}$ is not used when the node is in GATHERING_FACE_HUE_DATA state. The node is in this state only during the robot's initialization to gather face/background hue samples using the Viola and Jones (2001) face detector and the face/background region extraction heuristic described in section 3.3.1.3.

The finite-state machine that shows the transitions between each of the states of *rp_head_tracking* node is given in figure 4.8 and the behaviour of the node in each state is explained below, starting with a GATHERING_FACE_HUE_DATA state.


Gathered face hue sample count > n
DETECTING_HEADS
GATHERING_FACE_HUE_DATA
k frames elapsed since last detection, or |H_t|=0
|H_t|>0
TRACKING_HEADS
a) Node's state transitions if KLR skin hue classifier is used

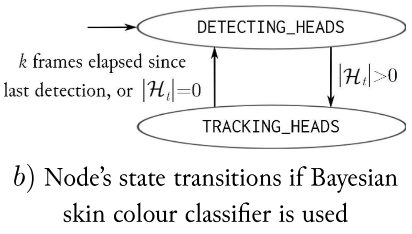

DETECTING_HEADS
k frames elapsed since last detection, or |H_t|=0
|H_t|>0
TRACKING_HEADS
b) Node's state transitions if Bayesian skin colour classifier is used

Figure 4.8: Finite-state machines of the *rp_head_tracker* ROS node if $a)$ the kernel logistic regression skin hue classifier, or $b)$ Bayesian skin colour classifier is used.

In the GATHERING_FACE_HUE_DATA state, frontal and profile face Viola and Jones (2001) detector cascades are used for face detection in incoming RGB frames until n face rectangles are detected[2]. Each of the detected face rectangles is split into face and background regions (as described in section 3.3.1.3), and two normalized hue histograms are constructed. This yields a set of training examples $\mathcal{D} = \{(\boldsymbol{h}_1, t_1), ..., (\boldsymbol{h}_{2n}, t_{2n})\}$ (where $\boldsymbol{h}_i$ is a normalized hue histogram, with $t_i = 1$ if the histogram is of skin region, and $t_i = 0$ otherwise). This training set is then used to train a kernel logistic regression (KLR) classifier.

To obtain a *maximum-a-posteriori* (MAP) estimate of the KLR's weight vector $\boldsymbol{\alpha}$, the objective function in equation (3.13) is minimized using the resilient backpropagation variant of the gradient descent algorithm (iRprop$^-$, Riedmiller (1994)). Since this algorithm requires a gradient of the likelihood function, it is calculated from equation (3.13) as $\nabla\mathcal{L}(\boldsymbol{\alpha}) = \left(\frac{\partial\mathcal{L}(\boldsymbol{\alpha})}{\partial\alpha_1}, ..., \frac{\partial\mathcal{L}(\boldsymbol{\alpha})}{\partial\alpha_n}\right)$, where

$$\begin{aligned}\frac{\partial}{\partial\alpha_k}\mathcal{L}(\boldsymbol{\alpha}) &= -\sum_i \left[\frac{\partial}{\partial\alpha_k} t_i \log\sigma(\textstyle\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i)) + \frac{\partial}{\partial\alpha_k}(1-t_i)\log(1-\sigma(\textstyle\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i)))\right] + \\ &+ \frac{\partial}{\partial\alpha_k}\frac{\sum_{i,j}\alpha_i\alpha_j K(\boldsymbol{h}_i, \boldsymbol{h}_j)}{2\lambda^2} \\ &= -\sum_i \left[t_i(1-\sigma(\textstyle\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i))) - (1-t_i)\sigma(\textstyle\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i))\right] K(\boldsymbol{h}_k, \boldsymbol{h}_i) + \\ &+ \frac{1}{\lambda^2}\sum_i K(\boldsymbol{h}_i, \boldsymbol{h}_k)\alpha_i \\ &= \sum_i \left[\frac{\alpha_i}{\lambda^2} + \sigma(\textstyle\sum_j \alpha_j K(\boldsymbol{h}_j, \boldsymbol{h}_i)) - t_i\right] K(\boldsymbol{h}_i, \boldsymbol{h}_k).\end{aligned}$$

[2]The implementation of the face detector with trained cascades is obtained from the open-source OpenCV library (Bradski, 2000).

Algorithm 4.2.3.1 A variant of the resilient backpropagation algorithm iRprop$^-$ (Riedmiller, 1994), as used by *rp_head_tracking* node for the kernel logistic regression classifier training. Given the objective function $\mathcal{L}(\boldsymbol{\alpha})$, its gradient $\nabla\mathcal{L}(\boldsymbol{\alpha})$, a kernel $K(\boldsymbol{x}, \boldsymbol{y})$, initial step sizes $\boldsymbol{s}$ and the time limit for training T this function attempts to find the weights $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$ minimizing $\mathcal{L}(\boldsymbol{\alpha})$.

```
RESILIENT-BACKPROPAGATION($\mathcal{L}(\boldsymbol{\alpha}), \nabla\mathcal{L}(\boldsymbol{\alpha}), \boldsymbol{s}, T$)
1   // Initialize the weights and the gradient for the next step
2   $\boldsymbol{\alpha} \leftarrow \mathbf{0}, \boldsymbol{g}' \leftarrow \mathbf{0}$

3   repeat
4       $\boldsymbol{g} \leftarrow \nabla\mathcal{L}(\boldsymbol{\alpha})$
5       for $i = 1$ to $n$
6           if $g_i \times g'_i > 0$ // Direction is unchanged
7               $s_i \leftarrow 1.2 s_i$
8           elseif $g_i \times g'_i < 0$ // Direction changed
9               $s_i \leftarrow 0.5 s_i$
10              $g_i \leftarrow 0$ // Force no change in the next iteration
11          // Update the weights based on step size and gradient direction
12          $\alpha_i \leftarrow \alpha_i - s_i \times \text{sign}(g_i)$
13      // Save the gradient
14      $\boldsymbol{g}' \leftarrow \boldsymbol{g}$
15  until time $T$ expires

16  return $\boldsymbol{\alpha}$
```

After obtaining the training example set $\mathcal{D}$ the *rp_head_tracking* node uses the adapted iRprop$^-$ algorithm (given in listing 4.2.3.1) to train a KLR classifier with an RBF (Gaussian) kernel (*i.e.* $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||^2}{2\delta^2}\right)$). A few examples of pattern classification using a kernel logistic regression trained using iRprop$^-$ algorithm are shown in figure 4.9.

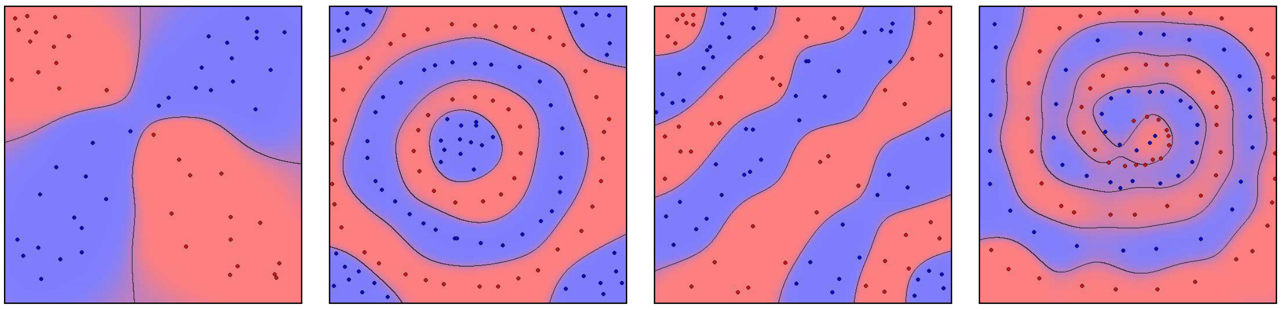


Figure 4.9: Sample non-linear patterns classified using logistic regression classifier with a radial basis function kernel and a Gaussian prior, trained using resilient backpropagation algorithm.

The visual summary of *rp_head_tracking* node's operation in `GATHERING_FACE_HUE_DATA` mode is shown in figure 4.10.

After training the kernel logistic regression classifier, the `rp_head_tracking` node switches into the `DETECTING_HEADS` state. In this state, an incoming depth image is searched for the presence of humans, and if there are any, the bounding rectangles of their heads are produced. To achieve this task, the extended Garstka and Peters' (2011) method (as described in section 3.3.1.2) is used to detect the candidate human head rectangles from the depth data.

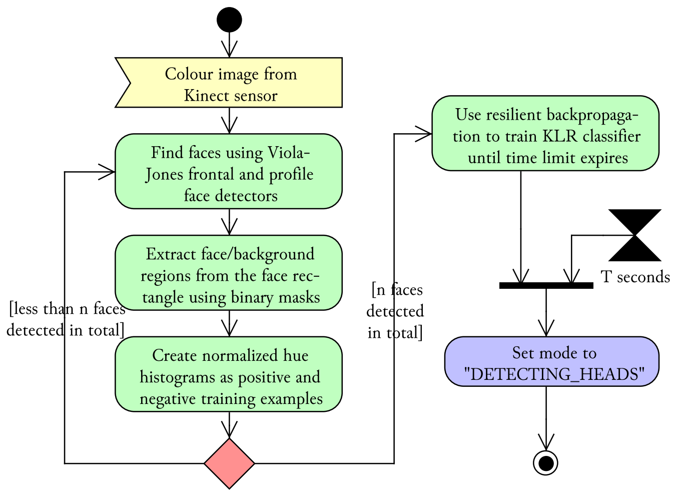

Colour image from Kinect sensor
Find faces using Viola-Jones frontal and profile face detectors
Extract face/background regions from the face rectangle using binary masks
Create normalized hue histograms as positive and negative training examples
[less than n faces detected in total]
[n faces detected in total]
Use resilient backpropagation to train KLR classifier until time limit expires
T seconds
Set mode to "DETECTING_HEADS"

Figure 4.10: A simplified UML activity diagram of the *rp_head_tracking* node's behaviour in `GATHERING_FACE_HUE_DATA` mode.

To incorporate the extra information present in the RGB data, these rectangles are further examined by either the KLR face hue classifier or a Bayesian skin colour classifier. If the KLR classifier is used then the binary mask from part c) of figure 3.51 is fitted to each of the detected head rectangles, and the hue histogram $\boldsymbol{h}$ is calculated from the region within the mask. This histogram is then normalized and used as a feature vector for the KLR classifier, which produces the probability $\Pr(skin|\boldsymbol{h}) = \sigma(\sum_i \alpha_i K(\boldsymbol{h}_i, \boldsymbol{h}))$. The overall detection is classified as a head if and only if $\Pr(skin|\boldsymbol{h}) > \theta$, where θ is a user-specified threshold.

If the Bayesian skin colour classifier is used (as described in detail in section 3.3.1.3) then the decision on whether the candidate head rectangle should be rejected/accepted is made based on the proportion of the skin pixels in the candidate rectangle. To count this proportion, a given pixel in the candidate head's window with values $R = r$, $G = g$ and $B = b$ (below shortened as rgb) is classified as belonging either to the skin, or non-skin region based on the likelihood ratio $\frac{\Pr(rgb|skin)}{\Pr(rgb|\neg skin)} \geq_? \theta$, where θ is a user-specified threshold.

The maximum likelihood estimates for the probabilities $\Pr(rgb|skin)$ and $\Pr(rgb|\neg skin)$ are learned off-line, from a large-scale, supervised Compaq skin-image dataset[3] containing 80,310,355 manually tagged skin and 861,173,849 non-skin pixels (Jones and Rehg, 2002).

After using the KLR or Bayesian classifiers to verify the head candidates, the final detected head set $\mathcal{R}_{t+1}$ is used to update the head history set $\mathcal{H}_{t+1}$ using the procedure described earlier. If $\mathcal{H}_{t+1} = \emptyset$, then the node stays in `DETECTING_HEADS` state and the head detection process is repeated for a new depth input frame. Otherwise, the *rp_head_tracking* node switches into `TRACKING_HEADS` state. This behaviour is visually summarized in figure 4.11.

In the `TRACKING_HEADS` state, a modified continuously adaptive mean-shift (CAMShift) algorithm is used to track detected heads using depth data (as described in detail in section 3.3.1.4). As a quick reminder, the depth image is first preprocessed by filtering depth shadows and smoothing it with an averaging filter. Then the constraints from equation (3.8) are used to reject local horizontal minima which could not possibly lie on the vertical head axis, thereby defining a degenerate head probability. The new head rectangle is obtained by finding the mode of this probability distribution using the mean-shift approach, while the size of the new rectangle is obtained from this distribution's zeroth moment, as described by Bradski (1998).

After obtaining the new head rectangle positions and sizes, degenerate rectangles (with zero area) are removed from

[3]This dataset was generously provided personally by M. Jones.

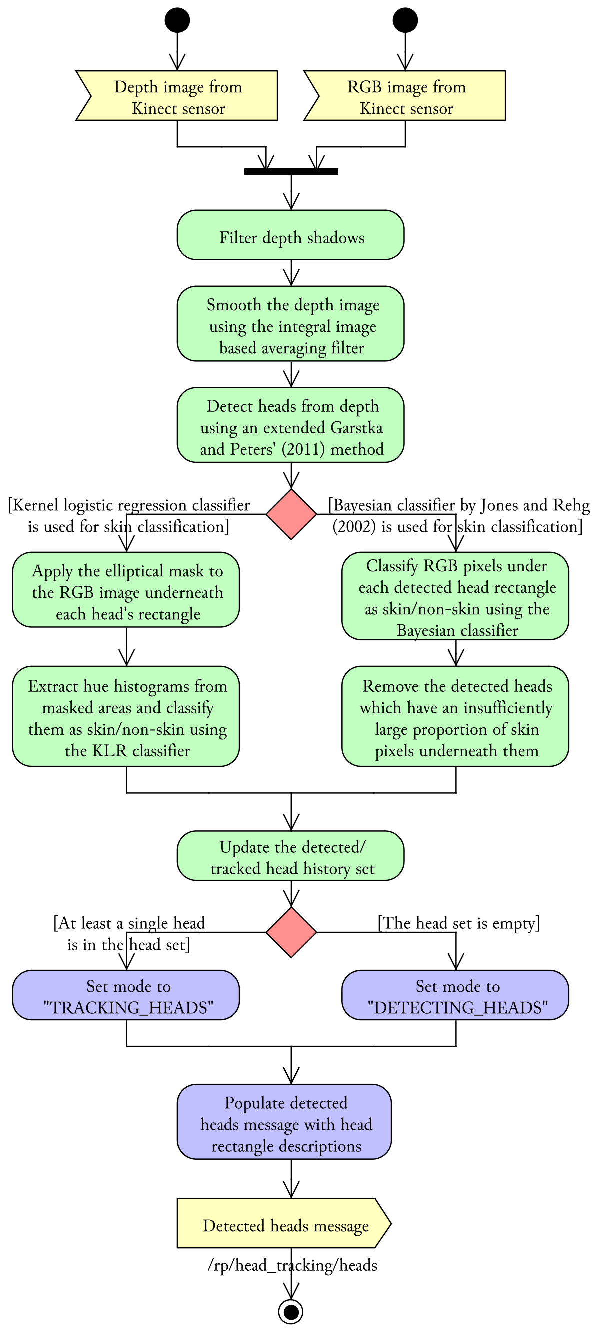

Depth image from Kinect sensor
RGB image from Kinect sensor
Filter depth shadows
Smooth the depth image using the integral image based averaging filter
Detect heads from depth using an extended Garstka and Peters' (2011) method
[Kernel logistic regression classifier is used for skin classification]
[Bayesian classifier by Jones and Rehg (2002) is used for skin classification]
Apply the elliptical mask to the RGB image underneath each head's rectangle
Classify RGB pixels under each detected head rectangle as skin/non-skin using the Bayesian classifier
Extract hue histograms from masked areas and classify them as skin/non-skin using the KLR classifier
Remove the detected heads which have an insufficiently large proportion of skin pixels underneath them
Update the detected/ tracked head history set
[At least a single head is in the head set]
[The head set is empty]
Set mode to "TRACKING_HEADS"
Set mode to "DETECTING_HEADS"
Populate detected heads message with head rectangle descriptions
Detected heads message
/rp/head_tracking/heads

Figure 4.11: A simplified UML activity diagram of the *rp_head_tracking* node's behaviour in `DETECTING_HEADS` mode.

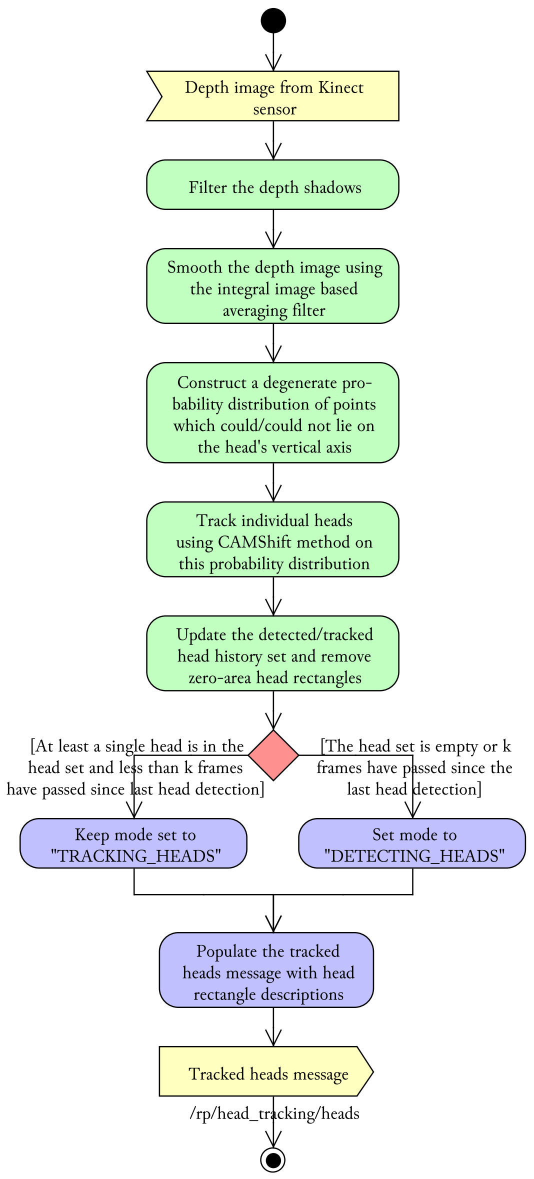
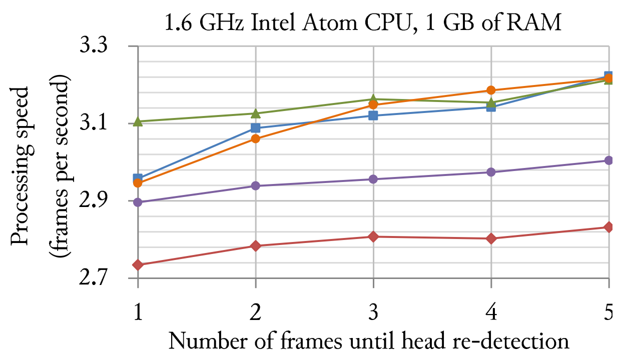

Depth image from Kinect sensor
Filter the depth shadows
Smooth the depth image using the integral image based averaging filter
Construct a degenerate probability distribution of points which could/could not lie on the head's vertical axis
Track individual heads using CAMShift method on this probability distribution
Update the detected/tracked head history set and remove zero-area head rectangles
[At least a single head is in the head set and less than k frames have passed since last head detection]
[The head set is empty or k frames have passed since the last head detection]
Keep mode set to "TRACKING_HEADS"
Set mode to "DETECTING_HEADS"
Populate the tracked heads message with head rectangle descriptions
Tracked heads message
/rp/head_tracking/heads

Figure 4.12: A simplified UML activity diagram of the *rp_head_tracking* node's behaviour in `TRACKING_HEADS` mode.

Legend	Depth shadow removal	Depth blurring ($r = 2$)	Verification using colour data
🟦	✗	✗	✗
🟥	✓	✗	✗
🟩	✓	✓	✗
🟣	✓	✓	Bayesian classifier
🟠	✓	✓	KLR classifier



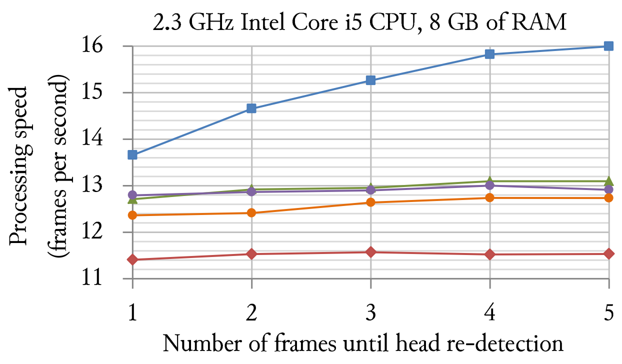


Figure 4.13: Performance evaluation of the head detection and tracking node (*rp_head_tracking*) with respect to different head re-detection thresholds k and various head detection/tracking modes.

the head history set. If the resulting head set is empty, *i.e.* $\mathcal{H}_{t+1} = \emptyset$ then the *rp_head_tracking* node immediately switches to `DETECTING_HEADS` mode. Otherwise, it stays in `TRACKING_HEADS` mode until k frames elapse since the last head detection attempt. The user-specified threshold k determines the trade-off between the computational efficiency and the node's responsiveness to new human subjects (*i.e.* a small k reduces the average time the robot needs to start tracking a new person in its FOV, and vice versa). This behaviour is again summarized visually in UML activity diagram 4.12.

The run-time performance of this node is measured for a variety of parameter combinations on both on-board and development computers. The results of this evaluation are presented in figure 4.13.

Photo composition and framing (rp_framing) The second most important node in Luke's "picture taking" behavioural capability layer is the photograph composition and framing (*rp_framing*) node. This node works as follows.

First of all, it subscribes to the locations of detected/tracked human subject heads in Kinect's image plane, published by the *rp_head_tracking* node. Then, this node maps the head locations from Kinect's image plane to the photographic camera's image plane and calculates the ideal framing based on the framing rules described by Dixon et al. (2003). If the calculated ideal frame lies outside the current photographic camera's image plane, a turn direction is proposed; otherwise, the ideal frame location is published over */rp/framing/frame* topic. These steps are described in more detail below.

In order to map the locations of detected heads from Kinect's to photographic camera's image plane, the following observations are used:

- Each point $\boldsymbol{p} = [x, y]^T$ on Kinect's image plane can be mapped to a corresponding point $\boldsymbol{P} = [X, Y, Z]^T$ in the world coordinates (with the Kinect sensor at the origin).
- The extrinsic transformation operator between Kinect sensor and photographic camera can be obtained by measuring the translation and rotation between the two devices on the robot's rigid frame. Let $\boldsymbol{t}$ be the obtained translation vector and $\boldsymbol{R}$ be the obtained rotation matrix.

- If the pinhole model of the camera is assumed, then the corresponding point $\boldsymbol{p}_c$ on the photographic camera's image plane could be calculated as

$$\boldsymbol{p}_c = \left[f_x \left(\frac{X_c}{Z_c} \right), f_y \left(\frac{Y_c}{Z_c} \right) \right]^T ,$$

where f_x and f_y are the horizontal/vertical focal lengths of the photographic camera and $\boldsymbol{P}_c = [X_c, Y_c, Z_c]^T$ are the coordinates of the world point $\boldsymbol{P}$ in the photographic camera reference frame, obtained by calculating

$$\boldsymbol{P}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \boldsymbol{RP} + \boldsymbol{t}. \tag{4.1}$$

However, the pinhole model is inaccurate due to $i)$ the principal point's displacement, and $ii)$ lens distortions. The former phenomenon occurs due to the camera CCD/CMOS sensor's center being slightly misaligned from the lens' optical axis. To deal with this inaccuracy, the model can be extended with two parameters c_x and c_y which represent the horizontal and vertical offsets of the image center coordinates from the camera's optical axis.

In this case, a world point projected onto the camera image plane would have the coordinates

$$\boldsymbol{p}_c = \left[f_x \left(\frac{X_c}{Z_c} \right) + c_x, f_y \left(\frac{Y_c}{Z_c} \right) + c_y \right]^T .$$

Define the camera intrinsics matrix as

$$\boldsymbol{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} , \tag{4.2}$$

then the projection of the world point $\boldsymbol{P}$ onto a camera image plane could be expressed in homogeneous coordinates as $\boldsymbol{p}_c = \boldsymbol{M}\boldsymbol{P}_c = \boldsymbol{M}(\boldsymbol{RP} + \boldsymbol{t})$.

To account for the second phenomena (*viz.* lens distortion) a ``plumb bob" model proposed by Brown (1966) can be used. This model simulates both radial distortion caused by the spherical shape of the lens, and the tangential distortion, arising from the inaccuracies of the assembly process.

More precisely, the radial distortion occurs since the light passing through the edges of the spherical lens is refracted more severely than the light passing through the center of the lens (as illustrated in parts $a)$ and $c)$ of figure 4.14). The tangential distortion, occurring when the lens and a CCD/CMOS sensor are not perfectly parallel is illustrated in parts $b)$ and $d)$ of figure 4.14.

In Brown's model the radial distortion is approximated by the Taylor series in the point's radial distance from the image plane's centre. Given an undistorted point on the camera's image plane $\boldsymbol{p}_i = [x_i, y_i]^T$, the distorted point $\boldsymbol{p}_d = [x_d, y_d]^T$ is defined as

$$\boldsymbol{p}_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix} = (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 + ...) \begin{bmatrix} x_i \\ y_i \end{bmatrix} , \tag{4.3}$$

where κ_i are the radial distortion parameters and $r = ||\boldsymbol{p}_i|| = \sqrt{{x_i}^2 + {y_i}^2}$.

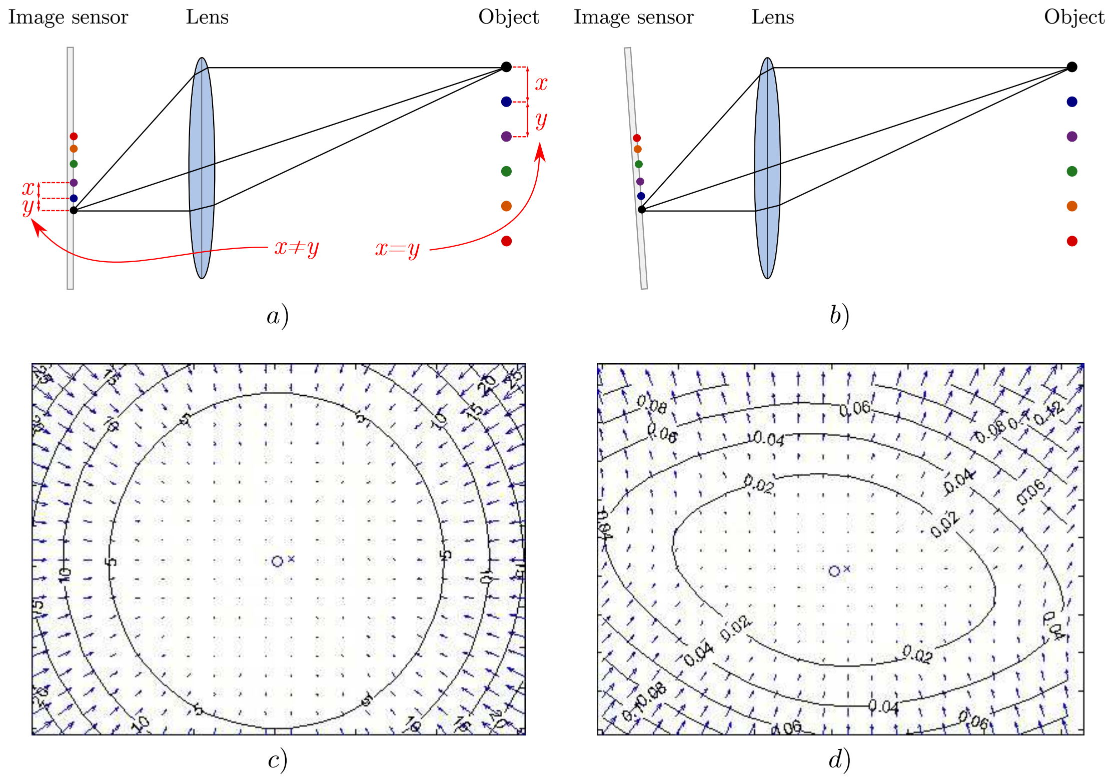


Figure 4.14: Example of radial and tangential lens distortions. Image $a)$ shows the radial (“barrel”) distortion, where rays further from the center of the lens are refracted stronger than the rays closer to the center (notice the equidistant object points on the right, and non-equidistant points on the sensor on the left). Image $b)$ shows the tangential distortion, occurring due to misalignment between the lens and the image sensor. Images $c)$ and $d)$ taken from Bouguet (2010) show the radial/tangential distortion plots for a particular lens, with the arrows indicating pixel displacements due to lens distortion.

The tangential distortion in Brown’s model is described as

$$\boldsymbol{p_d} = \begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} p_1(2x_iy_i) + p_2(r^2 + 2{x_i}^2) \\ p_2(2x_iy_i) + p_1(r^2 + 2{y_i}^2) \end{bmatrix}, \tag{4.4}$$

where p_1, p_2 are the tangential distortion parameters (see Brown (1966) for full derivation).

Then the final projection model which combines the principal point displacement and radial/tangential distortions can be defined by combining equations (4.1), (4.2), (4.3) and (4.4). In particular, let $\boldsymbol{P} = [X, Y, Z]^T$ be the point in Kinect-centred world coordinates. Then its projection on the photographic camera’s image plane $\boldsymbol{p}_c$ can be obtained in the following way:

1. Transform the point from world coordinates to the photographic camera’s coordinate frame using equation (4.1):
$$\boldsymbol{P}_c = \boldsymbol{RP} + \boldsymbol{t}.$$
2. Calculate the radial and tangential distortion in the dimensionless coordinates using equations (4.3) and (4.4). Let $[x_i, y_i]^T = \left[\frac{X_c}{Z_c}, \frac{Y_c}{Z_c}\right]^T$ and $r = \sqrt{{x_i}^2 + {y_i}^2}$. Then the distorted point $\boldsymbol{p_d}$ is given by
$$\boldsymbol{p_d} = \begin{bmatrix} x_d \\ y_d \end{bmatrix} = (1 + \kappa_1 r^2 + \kappa_2 r^4) \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} p_1(2x_iy_i) + p_2(r^2 + 2{x_i}^2) \\ p_2(2x_iy_i) + p_1(r^2 + 2{y_i}^2) \end{bmatrix},$$

where the Taylor series expansion up to the second order (κ_2) is used to approximate the radial distortion.

3. Finally, simulate the principal point displacement using the camera intrinsics matrix $\boldsymbol{M}$ from equation (4.2) to obtain the homogeneous coordinates of the projected point $\boldsymbol{p}_c$:

$$\boldsymbol{p}_c = \boldsymbol{M}\begin{bmatrix} \boldsymbol{p_d} \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} f_x x_d + c_x \\ f_y y_d + c_y \end{bmatrix}.$$

The parameters of this projection model (*viz.* $k_1, k_2, p_1, p_2, f_x, f_y, c_x$ and c_y) can be obtained automatically, using a set of calibration images. A common choice for such calibration objects is a chessboard pattern, as shown in figure 4.15.

Given the positions of the internal corners of the chessboard pattern in the image i and the prior knowledge that the points representing these corners should be strictly coplanar, the homography $\boldsymbol{H}_i$ between the object plane and the image plane can be found for each chessboard image i. Under the no-distortion assumption of the camera, the intrinsics matrix $\boldsymbol{M}$ can be calculated from the collected homography matrices $\boldsymbol{H}_i$ either using a closed-form solution (more prone to noise) or by the maximum-likelihood optimization using the Levenberg-Marquardt algorithm (see Zhang (1999, 2000) for full details).

Having obtained the intrinsic matrix containing focal length and principal point offset parameters, the distortion coefficients k_1, k_2, p_1, p_2 can be found using the method by Brown (1971). Essentially, the obtained intrinsic parameters are used to project the chessboard corner points onto the camera's image plane under the pinhole camera model. Assuming that the actual distorted locations of these points in the calibration images were produced by the the radial/tangential distortion model (equations (4.3) and (4.4)), the parameters k_1, k_2, p_1, p_2 can be calculated using a generalized least squares approach (by minimizing the sum of the squares of the distances between the distorted pinhole projections and the ground-truth positions in calibration images; see Brown (1971) for more details).

The implementations of Levenberg-Marquardt and Brown's approaches for the intrinsic matrix and distortion coefficient estimation respectively are provided in the open-source computer vision library (OpenCV, Bradski (2000)). Using this library, the point-and-shoot Nikon COOLPIX S3100 camera that Luke uses to take pictures is undistorted, obtaining an average chessboard corner point re-projection error of 0.658 pixels in 3.5 MP resolution calibration images (see figure 4.16 for the example of an undistorted image).

Using these obtained intrinsic and distortion parameters, the points from the depth point cloud provided by the Kinect sensor can be projected into the camera's image plane. Two example scenes showing the point clouds projected into the photographs taken with the Nikon COOLPIX S3100 camera are shown in figure 4.17.

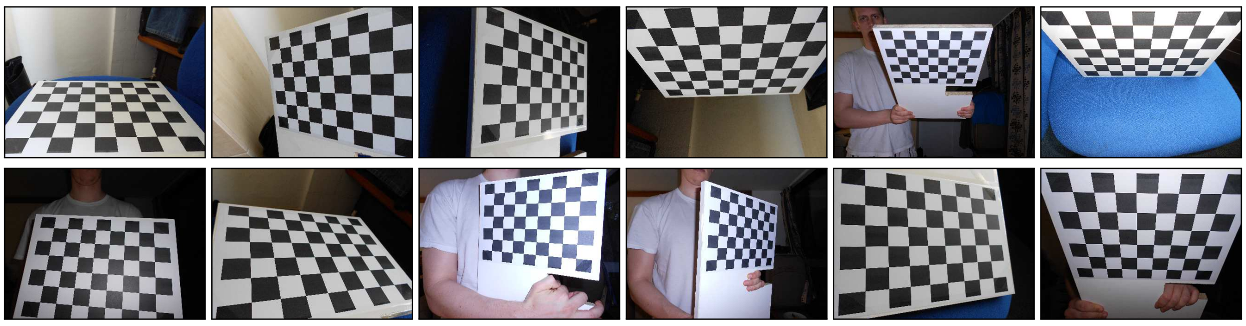


Figure 4.15: Sample calibration images ($4,320 \times 3,240$ pixel resolution) containing a chessboard pattern in various poses, used for the estimation of photographic camera's intrinsic matrix and distortion parameters.

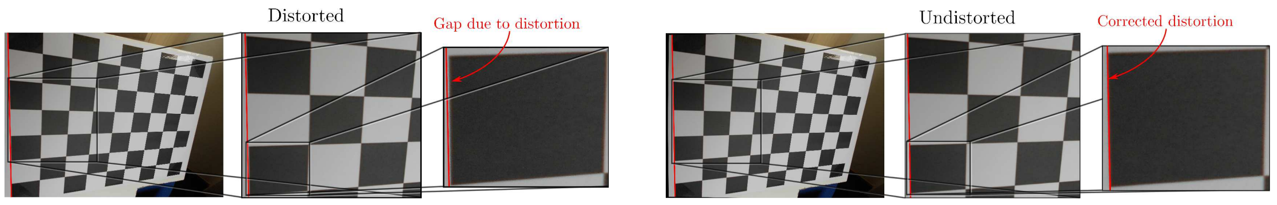


Figure 4.16: An example of an undistorted calibration image, obtained by inverting Brown's (1966) distortion model. Notice that the straight red line in the images on the left hand side does not go exactly through the corners of the chessboard due to tangential and radial distortion.

Using a similar approach, the 3D locations of the detected heads (provided by the *rp_head_tracking* node) are projected onto the photographic camera's image plane. Then, based on these locations the ideal framing for the photographs is calculated using the photograph composition heuristics proposed by Dixon et al. (2003). These heuristics are based on the following four photographic composition rules (Grill and Scanlon, 1990):

- *Rule of thirds*, which suggests that the points of interest in the scene should be placed at the intersections (or along) the lines which break the image into horizontal and vertical thirds.
- *No middle* rule, which states that a single subject should not be placed at a vertical middle line of the photograph.
- *No edge* rule, which states that the edges of an ideal frame should not be crossing through the human subjects.
- *Occupancy* ("empty space") rule, which suggests that approximately a third of the image should be occupied by the subject of the photograph.

Given these rules, Dixon et al. define three different heuristics for single person and wide/narrow group picture composition, illustrated in figure 4.18. In order to choose which heuristic will be used they employ an iterative procedure, which starts by identifying a human subject closest to the center of the current image. The ideal framing for this person is calculated using the single person composition heuristic from figure 4.18. If this frame includes

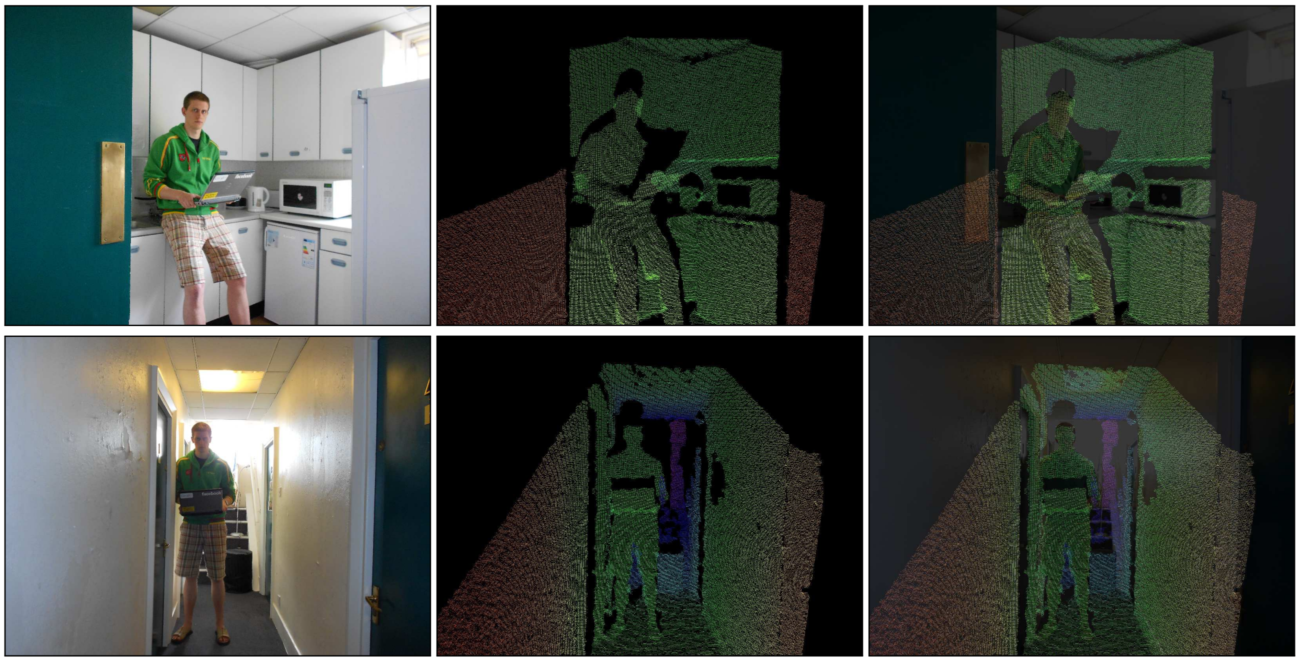


Figure 4.17: Two scenes with automatically merged RGB and depth data from the photographic camera and the Kinect sensor.

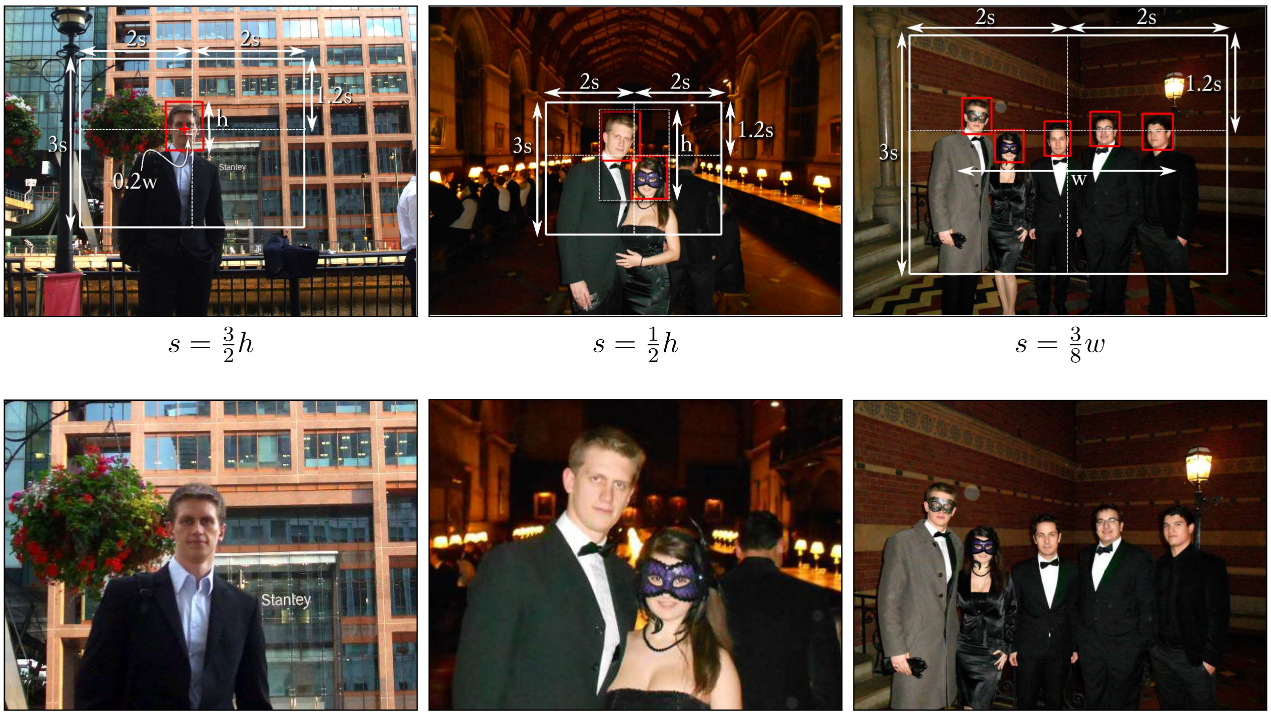
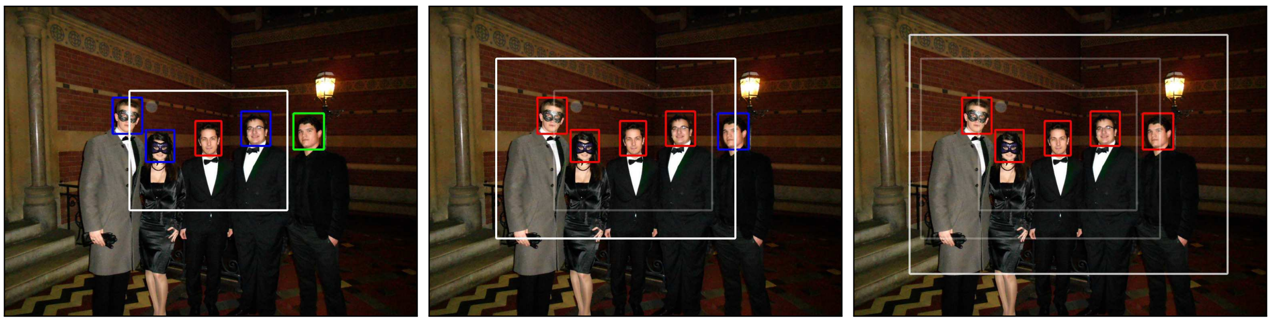


Figure 4.18: Ideal framing of single person and wide/narrow group shots, as proposed by Dixon et al. (2003). The composition rules for the single person/narrow group/wide group shots are shown in the top row of images (left-to-right). Images in the bottom row show the photos obtained from these proposed frames.

other candidate subjects, the group framing rules are applied iteratively on the expanded candidate set, until no new candidates are added. Figure 4.19 shows the intermediate frames obtained while composing a group picture using this procedure.

After an ideal frame F is calculated, the *rp_framing* node calculates the overlap coefficient O between the part of the frame visible in the current image I and the whole frame:

$$O = \frac{|I \cap F|}{|F|}.$$



a) Iteration 1 *b*) Iteration 2 *c*) Iteration 3 (final frame)

Figure 4.19: Illustration of the iterative framing method for the photo containing multiple human subjects, as described by Dixon et al. (2003). Detected human subjects which are considered at the current iteration are shown in red, subjects which need to be considered in the next iteration since they are present in the current "ideal frame" are shown in blue, and the subjects which are detected but can be ignored at the current iteration since they are outside the "ideal frame" are shown in green rectangles.

If the overlap coefficient O exceeds a given threshold θ_O and the visible part of the frame exceeds the minimal width/height thresholds $\theta_w \times \theta_h$, the node considers that the satisfying composition has been achieved and publishes the position/size of the ideal frame over the */rp/framing/frame* topic. Otherwise, the framing node determines the direction of where the robot should turn in order to improve the quality of the composition[4] and publishes these driving directions over the */rp/framing/driving_direction* topic.

In order to prevent the robot from getting stuck indefinitely while trying to achieve an ideal framing, a decaying temporal threshold for the minimum required overlap θ_O is used. In particular, let Δt be the time that the robot has already spent trying to frame the picture, $\max_{\Delta t}$ be the maximum allowed time for the picture framing and $\max_{\Delta O}$ be the maximum deviation from the ideal overlap that could still be tolerated. Then θ_O is defined as

$$\theta_O(\Delta t) = 1 - \min\left(\max_{\Delta O} \times \frac{\Delta t}{\max_{\Delta t}}, \max_{\Delta O}\right).$$

In the current robot photographer's implementation, the framing time limit $\max_{\Delta t}$ is set to 60 seconds, the maximum deviation from the ideal overlap $\max_{\Delta O}$ is set to 50% and the minimum visible frame size thresholds $\theta_w \times \theta_h$ are set to $2,160\ px \times 1,620\ px$.

The overall behaviour of *rp_framing* node is visually summarized in a simplified UML activity diagram in figure 4.20.

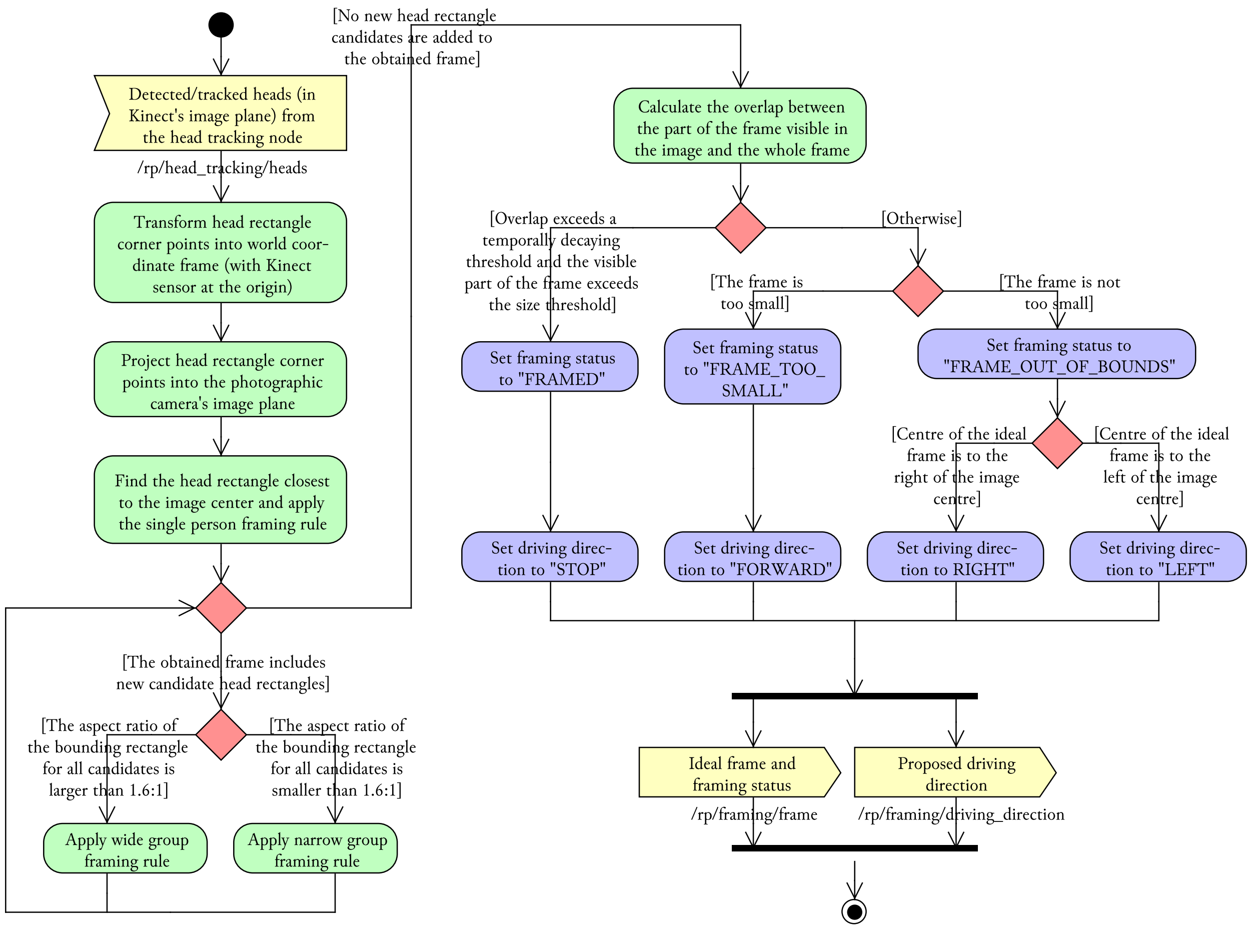

Detected/tracked heads (in Kinect's image plane) from the head tracking node
/rp/head_tracking/heads
Transform head rectangle corner points into world coordinate frame (with Kinect sensor at the origin)
Project head rectangle corner points into the photographic camera's image plane
Find the head rectangle closest to the image center and apply the single person framing rule
[No new head rectangle candidates are added to the obtained frame]
[The obtained frame includes new candidate head rectangles]
[The aspect ratio of the bounding rectangle for all candidates is larger than 1.6:1]
[The aspect ratio of the bounding rectangle for all candidates is smaller than 1.6:1]
Apply wide group framing rule
Apply narrow group framing rule
Calculate the overlap between the part of the frame visible in the image and the whole frame
[Overlap exceeds a temporally decaying threshold and the visible part of the frame exceeds the size threshold]
[Otherwise]
[The frame is too small]
[The frame is not too small]
Set framing status to "FRAMED"
Set framing status to "FRAME_TOO_SMALL"
Set framing status to "FRAME_OUT_OF_BOUNDS"
[Centre of the ideal frame is to the right of the image centre]
[Centre of the ideal frame is to the left of the image centre]
Set driving direction to "STOP"
Set driving direction to "FORWARD"
Set driving direction to RIGHT"
Set driving direction to "LEFT"
Ideal frame and framing status
/rp/framing/frame
Proposed driving direction
/rp/framing/driving_direction

Figure 4.20: A simplified UML activity diagram of the photograph composition and framing (*rp_framing*) ROS node.

[4]Based on the position of the ideal frame's centre *w.r.t.* the image's centre.

Autonomous photography process coordination (rp_autonomous_photography) The *rp_autonomous_photography* node coordinates the actual photograph taking/uploading process, and divides the robot's control time between the obstacle avoidance (*rp_obstacle_avoidance*) and framing (*rp_framing*) nodes.

More precisely, the behaviour of the *rp_autonomous_photography* node is split into individual cycles, where at the end of each cycle a new, well-composed picture is taken. Each cycle divided into two halves: in the first half the autonomous photography node instructs *rp_navigation* node to use the driving direction inputs from the obstacle detection and avoidance node. This allows the robot to randomly navigate the environment for a fixed amount of time, and ensures that the robot does not take all pictures from the same position.

After k seconds (where k is the duration of the obstacle avoidance phase, settable at launch or through the ROS parameter server) the autonomous photography node gives control to the framing node, which attempts to produce an aesthetically pleasing picture composition. This is the second half of the picture taking cycle.

As soon as the framing node establishes a satisfactory framing (indicating it by publishing "`FRAMED`" state message), the autonomous photography node issues a photo request to *rp_camera* node. Once the picture is taken and downloaded from the camera to an on-board computer, the autonomous photography node crops out the proposed frame rectangle and uploads it to Flickr using *rp_uploader* node. After uploading the cropped-out frame, the autonomous photography node gives the control back to the obstacle avoidance node and a new photograph-taking cycle is started.

This behaviour is illustrated visually in UML activity diagram in figure 4.21.

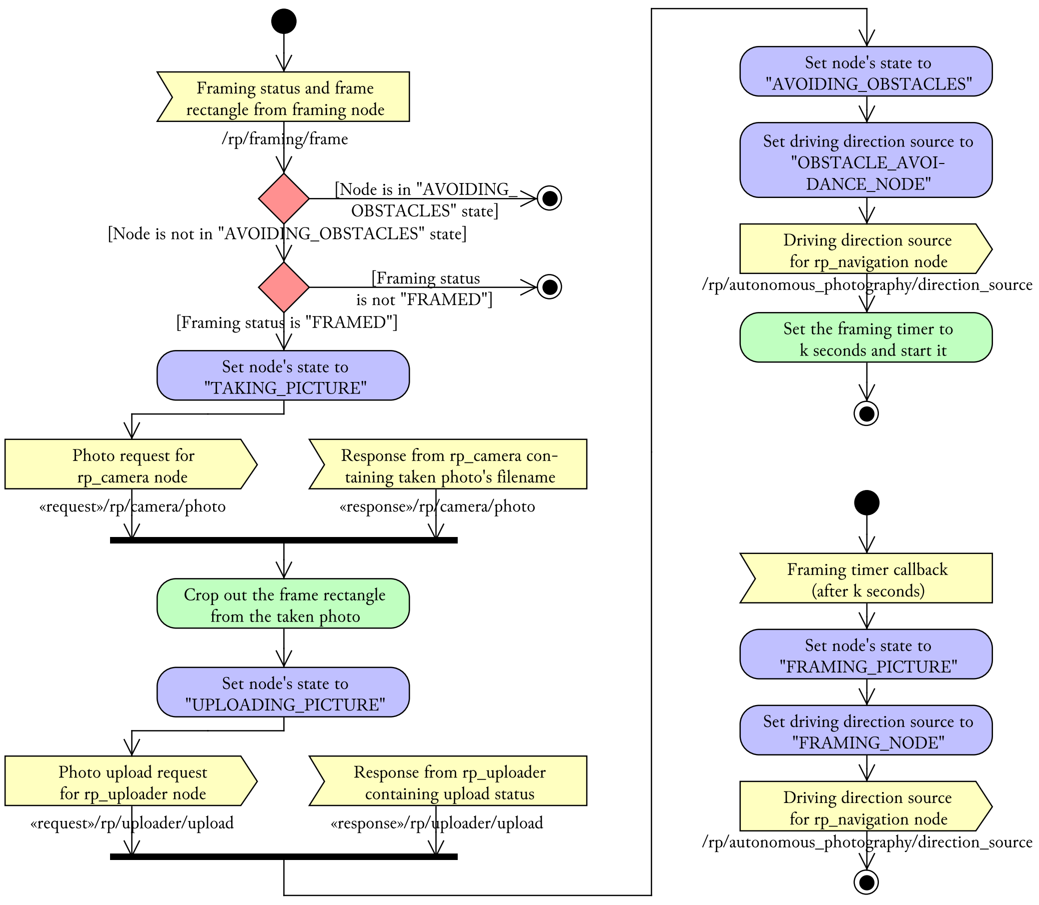

Framing status and frame rectangle from framing node
/rp/framing/frame
[Node is in "AVOIDING_OBSTACLES" state]
[Node is not in "AVOIDING_OBSTACLES" state]
[Framing status is not "FRAMED"]
[Framing status is "FRAMED"]
Set node's state to "TAKING_PICTURE"
Photo request for rp_camera node
«request»/rp/camera/photo
Response from rp_camera containing taken photo's filename
«response»/rp/camera/photo
Crop out the frame rectangle from the taken photo
Set node's state to "UPLOADING_PICTURE"
Photo upload request for rp_uploader node
«request»/rp/uploader/upload
Response from rp_uploader containing upload status
«response»/rp/uploader/upload
Set node's state to "AVOIDING_OBSTACLES"
Set driving direction source to "OBSTACLE_AVOIDANCE_NODE"
Driving direction source for rp_navigation node
/rp/autonomous_photography/direction_source
Set the framing timer to k seconds and start it
Framing timer callback (after k seconds)
Set node's state to "FRAMING_PICTURE"
Set driving direction source to "FRAMING_NODE"
Driving direction source for rp_navigation node
/rp/autonomous_photography/direction_source

Figure 4.21: A simplified UML activity diagram of the autonomous photography process coordination (*rp_autonomous_photography*) ROS node.

Photograph taking (rp_camera) The photograph taking node (*rp_camera*) acts as an interface between other ROS nodes and the physical Nikon COOLPIX S3100 camera that Luke uses to take pictures.

To communicate with the photographic camera this node uses the *libgphoto2* API for the open-source gPhoto^2 (Barbé et al., 2002) library, which in turn connects to the camera using the Picture Transfer Protocol (PTP). This node provides access to the camera for the rest of the Luke's ROS graph by exposing a ROS service at */rp/camera/photo*. Any other ROS node can send an empty request to this service, which *rp_camera* node transforms into the photo capture request for the *libgphoto2* API. This request triggers a physical camera capture, storing the taken picture in the camera's built-in memory. After the picture is taken, *rp_camera* node moves the picture from the camera's memory to the on-board computer and returns the string file name of the downloaded picture via the service response. This basic behaviour of *rp_camera* node is visually summarized in 4.22.

This node also exposes a couple of parameters (settable either at launch time or at run-time via the parameter server), which allow other nodes to modify basic camera parameters, like the flash mode. By default, the flash is set to fire for every picture, acting as an additional visual cue of the picture taking moment. Also, while this node can be used more broadly, at the moment only the *rp_autonomous_photography* node uses it to take pictures of human subjects in Luke's environment.

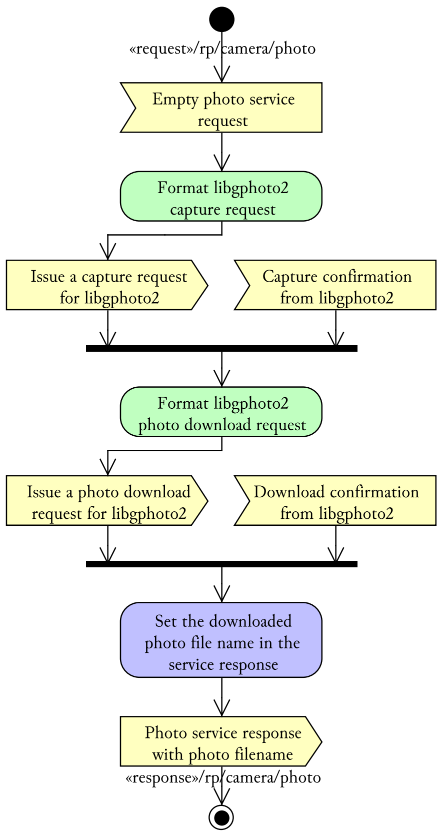

«request»/rp/camera/photo
Empty photo service request
Format libgphoto2 capture request
Issue a capture request for libgphoto2
Capture confirmation from libgphoto2
Format libgphoto2 photo download request
Issue a photo download request for libgphoto2
Download confirmation from libgphoto2
Set the downloaded photo file name in the service response
Photo service response with photo filename
«response»/rp/camera/photo

Figure 4.22: A simplified UML activity diagram of the photograph taking (*rp_camera*) ROS node.

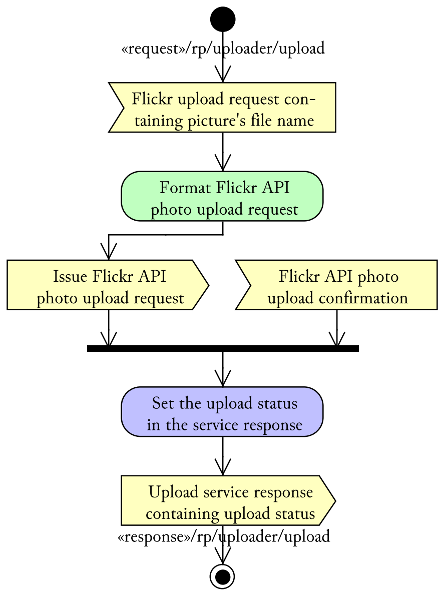

«request»/rp/uploader/upload
Flickr upload request containing picture's file name
Format Flickr API photo upload request
Issue Flickr API photo upload request
Flickr API photo upload confirmation
Set the upload status in the service response
Upload service response containing upload status
«response»/rp/uploader/upload

Figure 4.23: A simplified UML activity diagram of the photograph uploading (*rp_uploader*) ROS node.

Photograph uploading (rp_uploader) The photograph uploading node (*rp_uploader*) uses the Python Flickr API (Stüvel, 2007) to upload image files to an online Flickr photo gallery. It exposes the Flickr API to the rest of ROS graph by providing */rp/uploader/upload* ROS service.

Any node in Luke's node graph can send a ROS request to this service containing a file name of the picture to upload. After receiving this request the *rp_uploader* node loads the picture from the hard drive of an on-board computer and uploads it to Flickr; the upload status (success/fail) is returned to the calling node as the service response (as illustrated in figure 4.23). Other parameters of the uploaded photo (like title, description or tags) can be set through the ROS parameter server.

At the moment, only the *rp_autonomous_photography* node uses the *rp_uploader* node to upload the taken photographs.

The internet connection required for the picture uploading is provided by the on-board HTC HD7 phone (which also acts as a robot's state display) by tethering the phone's 3G/EDGE data connection over Wi-Fi to an on-board netbook which runs the overall Luke's ROS graph.

4.2.3.3 Externalization of the current state via vocal and visual messages

The third and final behavioural competence of the implemented autonomous photographer robot involves its ability to externalize the current state via synthesized voice messages (played over the on-board computer's speakers), and text messages/QR codes (shown on the display of the attached HTC HD7 phone). This competence layer is implemented by three basic ROS nodes (*rp_state_externalization*, *rp_speech* and *rp_display*) and a Windows Phone 7.8 app which is running on the attached phone. Each of these software components is briefly discussed below.

State externalization (rp_state_externalization) The state externalization node (*rp_state_externalization*) subscribes to the status outputs from all major nodes in Luke's ROS graph, in particular, the lomotion (*rp_locomotion*), head tracking (*rp_head_tracking*), framing (*rp_framing*) and photography process control (*rp_autonomous_photography*) nodes.

Each of these nodes can be in one of the following states:

$$S_{locomotion} \in \{\texttt{NORMAL}, \texttt{PROCESSING\_BUMPER\_EVENT}\},$$
$$S_{head\ tracking} \in \{\texttt{GATHERING\_FACE\_HUE\_DATA}, \texttt{DETECTING\_HEADS}, \texttt{TRACKING\_HEADS}\},$$
$$S_{framing} \in \{\texttt{NO\_FRAME}, \texttt{FRAME\_OUT\_OF\_BOUNDS}, \texttt{FRAME\_TOO\_SMALL}, \texttt{FRAMED}\},$$
$$S_{auton.\ photography} \in \{\texttt{AVOIDING\_OBSTACLES}, \texttt{TAKING\_PICTURE}, \texttt{UPLOADING\_PICTURE}, \texttt{FRAMING\_PICTURE}\}.$$

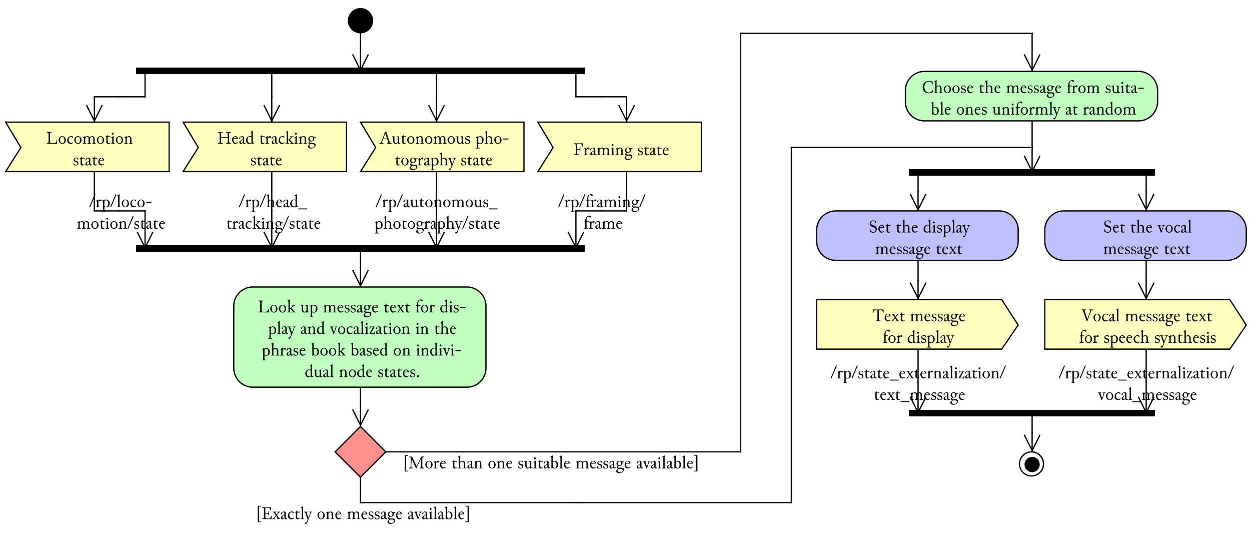
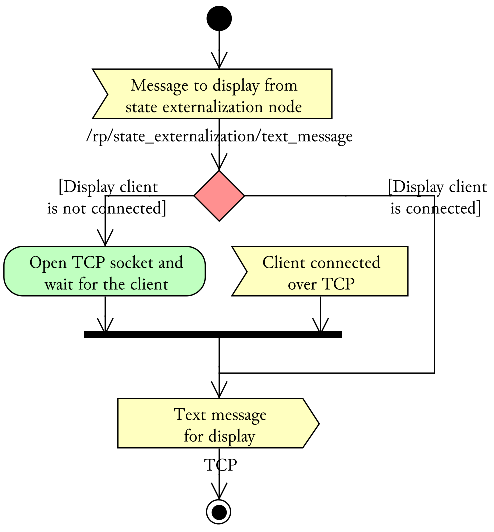

Locomotion state
Head tracking state
Autonomous photography state
Framing state
/rp/locomotion/state
/rp/head_tracking/state
/rp/autonomous_photography/state
/rp/framing/frame
Look up message text for display and vocalization in the phrase book based on individual node states.
[More than one suitable message available]
[Exactly one message available]
Choose the message from suitable ones uniformly at random
Set the display message text
Set the vocal message text
Text message for display
Vocal message text for speech synthesis
/rp/state_externalization/text_message
/rp/state_externalization/vocal_message

Figure 4.24: A simplified UML activity diagram of the state externalization (*rp_state_externalization*) ROS node.

In order to produce the robot's state messages (which are later vocalized/displayed by *rp_speech* and *rp_display* nodes) the state externalization node uses a table of pre-defined text messages, indexed by tuples $(S_{locomotion}, S_{head\ tracking}, S_{framing}, S_{auton.\ photography})$. If the table contains more than one message for a given tuple, then the message to be produced is chosen uniformly at random from the matching messages. An excerpt from this table is shown in listing A.1 in appendix A.1.

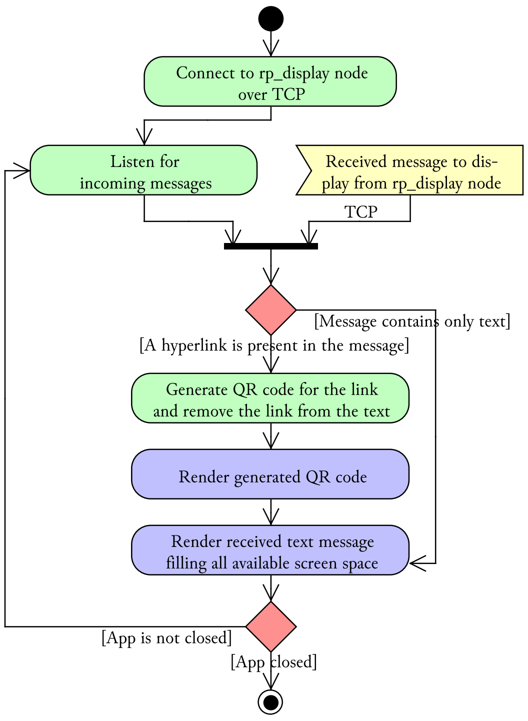
After the message texts are chosen for the display/synthesized audio messages, they are published via */rp/state_externalization/text_message* and */rp/state_externalization/vocal_message* topics (as illustrated in figure 4.24). The nodes responsible for message display and vocalization subscribe to these topics and produce the appropriate outputs, as described below.

Message display (rp_display) The message display node (*rp_display*) acts as a proxy between the state externalization node and the display attached to the robot's frame. To provide OS and device independence for the physical display, this node simply creates a TCP server and sends new messages received from the state externalization node to the TCP client (as illustrated in figure 4.25).



Message to display from state externalization node
/rp/state_externalization/text_message
[Display client is not connected]
[Display client is connected]
Open TCP socket and wait for the client
Client connected over TCP
Text message for display
TCP

Figure 4.25: A simplified UML activity diagram of the message display (*rp_display*) ROS node.

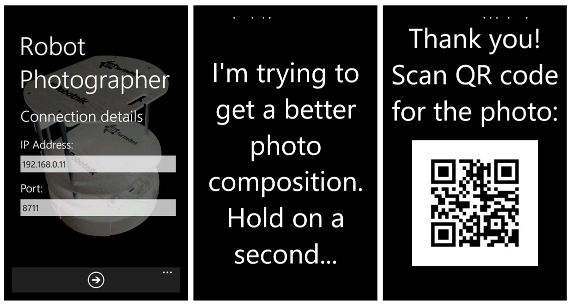


Connect to rp_display node over TCP
Listen for incoming messages
Received message to display from rp_display node
TCP
[Message contains only text]
[A hyperlink is present in the message]
Generate QR code for the link and remove the link from the text
Render generated QR code
Render received text message filling all available screen space
[App is not closed]
[App closed]

Figure 4.26: A simplified UML activity diagram of message display Windows Phone 7.8 app.

In the current implementation, an HTC HD7 phone is used to show the received messages. This phone has a 4.3 inch, 480 × 800 pixel LCD display, and is running Windows Phone (WP) 7.8 operating system. To show the messages generated by *rp_state_externalization* node, a basic message display WP OS app is written. Essentially this app connects to the *rp_display* node over TCP and renders received text messages in full-screen mode. If a hyperlink is present within the received text message then this app also generates and renders a QR (Quick Response) code. This makes it easier for the humans in the robot's vicinity to follow this link, since any modern phone can use the phone's camera to automatically read QR codes.

A couple of screenshots of this app are shown in figure 4.27 and its basic UML activity diagram is shown in figure 4.26.



Robot Photographer
Connection details
IP Address:
192.168.0.11
Port:
8711

I'm trying to get a better photo composition. Hold on a second...

Thank you! Scan QR code for the photo:

a) *b*) *c*)

Figure 4.27: Sample screenshots of the robot's display app for Windows Phone 7.8. Image *a*) shows the configuration screen which allows this app to connect to the *rp_display* node, image *b*) shows the received and rendered text message, and image *c*) shows both the received text message and the rendered QR hyperlink code.

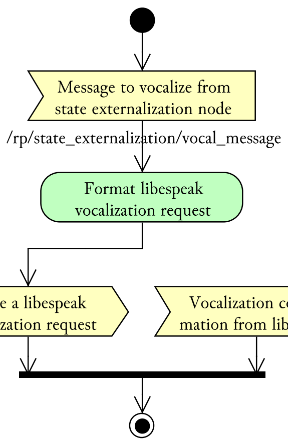

Message to vocalize from state externalization node
/rp/state_externalization/vocal_message
Format libespeak vocalization request
Issue a libespeak vocalization request
Vocalization confirmation from libespeak

Figure 4.28: A simplified UML activity diagram of the message vocalization (*rp_speech*) ROS node.

Message vocalization (rp_speech) To vocalize the text messages sent by *rp_externalization* node, the *rp_speech* node uses an open-source eSpeak (Duddington, 2006) speech synthesis engine, which in turn is configured to use a formant[5] synthesis based approach as described by Klatt (1980).

In essence, Klatt's voice synthesizer works in two stages: first of all, two parallel harmonic signals are produced, which simulate the vibration of the vocal chords. Secondly, a cascade of low pass, resonance/anti-resonance, additive and other digital filters is applied to simulate the rest of the vocal tract transfer function, such that the resulting waveform resembles human speech (see Klatt (1980) for more details). Since this method does not need a database of speech samples and uses computationally cheap digital signal filters, the resulting text-to-speech engine is both memory and CPU efficient, making it highly appropriate for the use in a mobile robot.

The actual *rp_speech* node acts as a thin wrapper between the eSpeak engine and the rest of Luke's ROS graph (although in the current implementation only the *rp_state_externalization* uses this node to generate vocal outputs). In particular, the *rp_speech* node simply subscribes to the messages published in */rp/state_externalization/vocal_message* topic and forwards them to the *libespeak* API, as visually summarized in figure 4.28.

4.2.4 Computational resource usage of individual nodes

In order to illustrate how CPU and memory resources are used by the individual Luke's nodes, their usage statistics were gathered over a ten minute sequence of Luke's operation. These cumulative statistics are presented in figures 4.29 and 4.30.

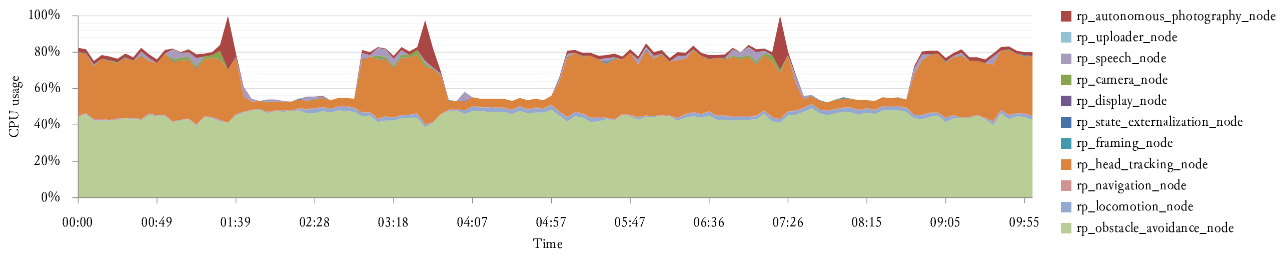


Figure 4.29: Cumulative CPU usage of Luke's ROS nodes over the ten minute test deployment.

[5]Fant (1960) defines formants as "the spectral peaks of the sound spectrum of the voice".

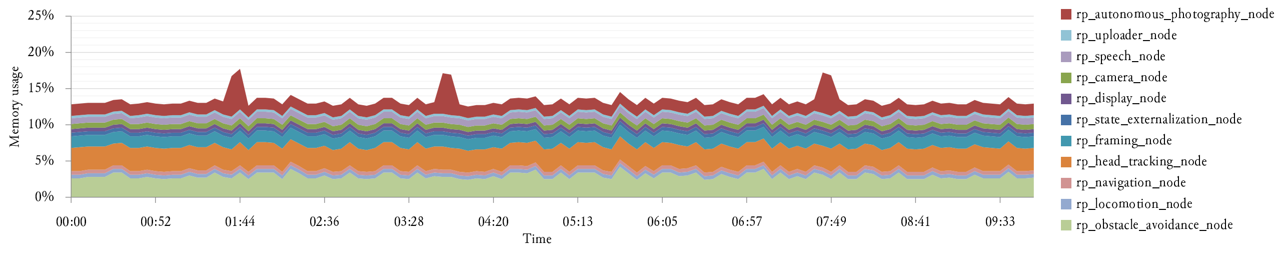


Figure 4.30: Cumulative memory usage of Luke's ROS nodes over the ten minute test deployment.

As can be seen from these charts, obstacle detection/avoidance (*rp_obstacle_avoidance*) and head detection/tracking (*rp_head_tracking*) nodes use the largest amount of the computational resources. Furthermore, the "photograph framing"/"random wandering" cycles can also be clearly seen, especially from the CPU usage diagram, since during the "random wandering" cycle the head detection/tracking node is disabled. The memory usage diagram also illustrates that the memory usage remains relatively constant over this time period, indicating that the nodes do not have memory leaks.

The qualitative performance of the robot photographer is described in the following chapter, which discusses Luke's deployment in a real-world event and thoroughly evaluates the quality of the pictures taken.

Chapter 5

Insights from Robot Photographer's Deployment in Real-World

This chapter summarizes the experiences from Luke's deployment in an unstructured real-world event and provides statistical evaluation of the pictures taken. The obtained results are compared with the earlier autonomous robot photographer approaches.

5.1 Autonomous robot photographer deployment at an open-day event

In June 2013 the robot has been deployed at an open-day event for prospective CS undergraduate students in the Department of Computer Science, University of Oxford. The event was running for two consecutive days and was attended by more than four hundred prospective students, parents and teachers.

The programme of each day was divided into multiple half-an-hour talks, interview and Q&A sessions, given in lecture theatres and computing rooms. During the occasional fifteen minute breaks between these events, the attendees would come out and mingle in the main "atrium" area (shown in figure 5.1). The robot photographer would be active during these breaks, autonomously taking pictures in the unstructured environment.

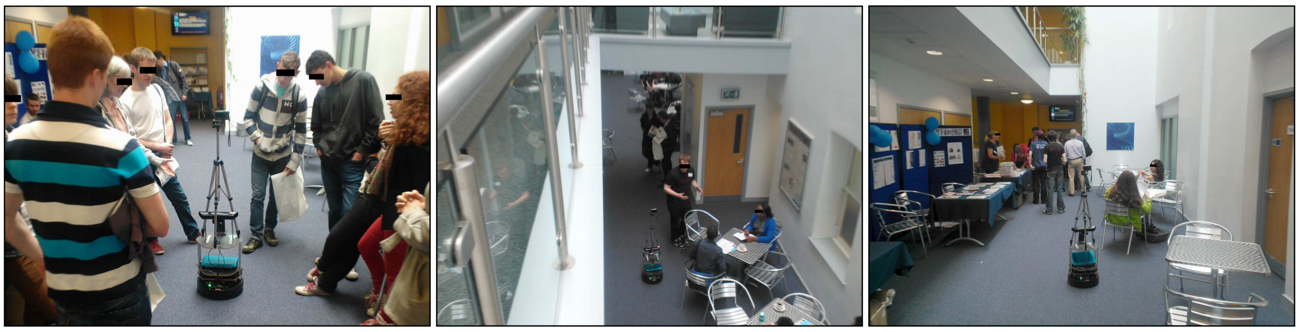


Figure 5.1: An autonomous robot photographer Luke "in-action" at the open day event in the Department of Computer Science, University of Oxford.

For its successful operation, Luke had to avoid both static obstacles (chairs, tables, presentation stands, walls, *etc.*) and dynamic obstacles (*viz.* attendees, randomly milling about). In order to ensure Luke's safe operation, its linear velocity was limited to 10 cm/s, and the angular velocity was limited to 0.5 rad/s. At this speed, the obstacle detection and avoidance method described in sections 3.3.2 and 4.2.3.1 performed flawlessly, allowing Luke to avoid any collisions during its operation. The only times when human supervision was required were when Luke was about to wander out into the corridors leading away from the atrium. In these cases Luke was directed back to the atrium by blocking its path and forcing him to turn around; such interventions were required around once in every thirty minutes of Luke's operation.

Due to its relatively slow speed (and noisiness of the environment, which was burying Luke's audio messages), the robot often was able to take candid photographs of the attendees while remaining unnoticed (just like a human event photographer would). However, some attendees (particularly older parents) indicated that they found the presence of the robot somewhat unsettling or creepy because of its ability to track people and navigate the environment autonomously. This could have been caused by the fact that Luke has a very limited set of reactive interaction skills with humans: most of the times the HRI was limited to the attendees blocking the robot's path, and Luke changing his driving direction to avoid them. At this point, the attendees would notice the message on Luke's status display that he is "looking around for good picture locations" (or similar), and leave him alone.

In order to detect and track people based on RGB-D data, Luke was using the methods described in sections 3.3.1 and 4.2.3.2. After some brief initial testing of the KLR and Bayesian skin classifiers (described in section 3.3.1.3), the latter was found to be performing better in this particular environment and was used for the remainder of Luke's two-day deployment.

Over the total ten and a half hour period of Luke's presence at the open-days, the robot spent three hours and thirty-seven minutes actively taking pictures (*viz.* during the breaks between various events). In total Luke took 103 pictures, approximately one picture every two minutes. The breakdown of these statistics for each day is shown in table 5.1.

Day	Total robot's presence time[1]	Total duration of robot's activity[1]	Number of photos taken	Average time b/t pictures[1]
Day 1	05:26:25	02:11:34	57	00:02:18
Day 2	05:10:16	01:25:59	46	00:01:52
Total	10:36:41	03:37:33	103	00:02:07

[1] In *hh:mm:ss* format.

Table 5.1: Statistics of Luke's deployment at an open-day event.

The quality of the pictures was quantitatively evaluated on a five-point Likert scale (Likert, 1932), and compared to the results obtained by earlier robot photographers, as presented in the following section.

5.1.1 Evaluation of pictures taken by the robot

In order to quantitatively evaluate the quality of the pictures taken by Luke, sixteen people (unrelated to the project) were asked to evaluate all 103 pictures using an on-line photo rating tool, developed specifically for this project. This tool, and the methodology of the rating collection experiment is described below.

5.1.1.1 On-line rating tool for fast photo rating

To streamline the process of photograph rating, an online tool was implemented (shown in figure 5.2). This tool enabled participants to rate and navigate through Luke's photos using simple keyboard shortcuts[1].

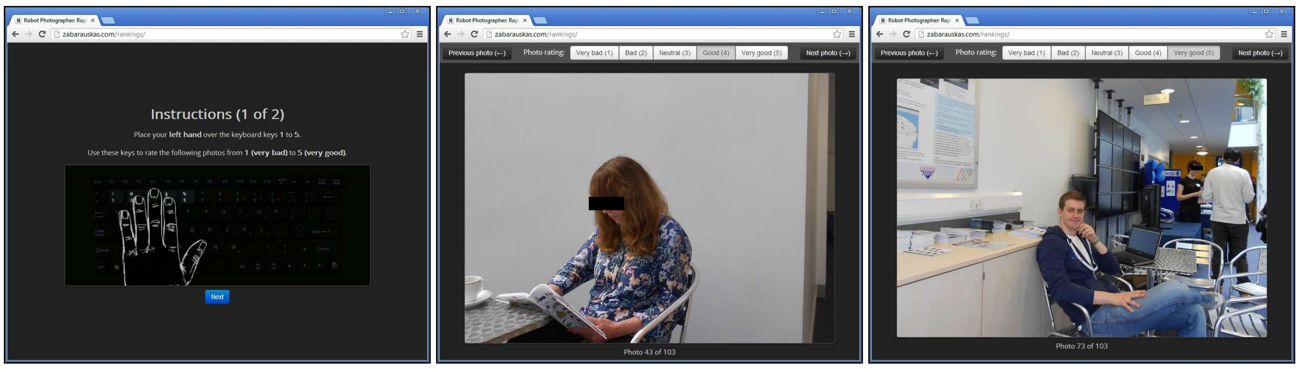
When participants opened the photo rating web page, the tool displayed two animations explaining the keyboard shortcuts for assigning photo ratings (*viz.* keyboard keys "1"–"5") and the shortcuts for navigation between pictures (*viz.* left-arrow key "←" to go back to the previous picture and right-arrow key "→" to proceed to the next picture after leaving a rating).

After this introduction the participants were asked to rate each picture that Luke took on the following Likert scale:

Very bad (1), *Bad (2)*, *Neutral (3)*, *Good (4)*, *Very good (5)*.

To reduce the combined perceptual correlations between the neighbouring photographs, all pictures were shown in a randomized order for each participant of the experiment.

[1]Obviously, participants could also use the mouse to perform the same actions.


Instructions (1 of 2)

Place your **left hand** over the keyboard keys **1** to **5**.

Use these keys to rate the following photos from **1 (very bad)** to **5 (very good)**.

[redacted]

[redacted]

Figure 5.2: Screenshots of the developed tool for photo rating on the five-point Likert scale.

The tool itself was developed using an industry-standard LAMP stack (Linux OS, Apache HTTP server, MySQL DBMS and PHP programming language). The front-end UI was powered by the open-source Bootstrap framework (Twitter, 2013) written in JavaScript, CSS and HTML. A MySQL database was used for the storage of ratings in the back-end, and the middle layer connecting these components was programmed using PHP.

Using this tool a set of 1,648 ratings was collected (*i.e.* sixteen ratings on the five-point Likert scale for each picture).

These ratings are thoroughly analysed and compared to the results obtained by other autonomous robot photographer approaches in the following sections, starting with the examination of reliability of the collected ratings, described below.

5.1.1.2 Inter-rater agreement measurement

To better understand how reliable are the collected ratings, two statistical methods were applied: the weighted Cohen's kappa statistic (Cohen, 1968), and a simple percentage agreement. Each of these methods are briefly described below.

Weighted Cohen's kappa To measure the inter-rater agreement while discounting the agreement between raters that occurs simply by chance, a weighted Cohen's kappa statistic can be used. This statistic is calculated in the following way.

Let $\boldsymbol{X} = [x_{i,j}]$ be the matrix of observed disagreements between the two judges, where $x_{i,j}$ counts the number of times when the first judge gave the rating i and the second judge gave the rating j. Furthermore, let $\boldsymbol{W} = [w_{i,j}]$ be the quadratic penalty matrix for the disagreement when the first judge gives rating i and the second judge gives rating j, with $w_{i,j} = (i - j)^2$.

As described by Cohen, this allows the ratings on the Likert scale to be treated as ordinal data (*i.e.* without the assumption that they are spaced equidistantly), while at the same time accounting for partial disagreements between the judges on the same picture.

In order to take into consideration the amount of agreement between the judges which would be expected by chance, let $\hat{\boldsymbol{x}}_i = [\hat{x}_{i,r}]^T$ be the vector of cumulative counts when judge i gave rating r. Then the expected disagreement matrix $\boldsymbol{M} = [m_{i,j}]$ can be defined as

$$m_{i,j} = \frac{\hat{x}_{1,i}}{||\hat{\boldsymbol{x}}_1||_1} \times \hat{x}_{2,j} = \frac{\hat{x}_{1,i} \times \hat{x}_{2,j}}{103},$$

where $||\hat{\boldsymbol{x}}_1||_1 = \sum_{i=1}^{5} \hat{x}_{1,i} = 103$ is an L_1 norm of the cumulative rating count vector $\hat{\boldsymbol{x}}_1$ for the total of 103 pictures.

Finally, the weighted Cohen's kappa statistic can be calculated as

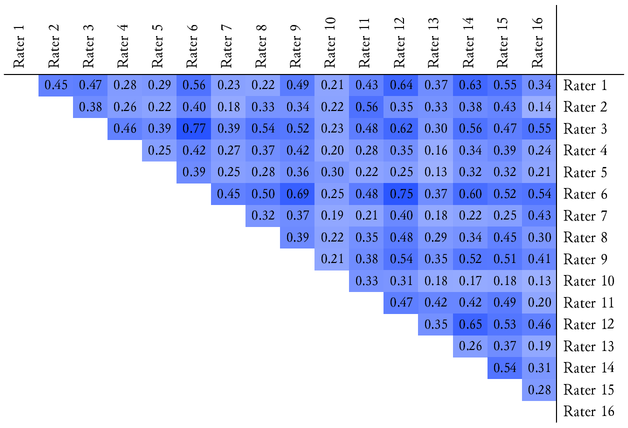
$$\kappa = 1 - \frac{\sum_{i=1}^{5} \sum_{j=1}^{5} x_{i,j} w_{i,j}}{\sum_{i=1}^{5} \sum_{j=1}^{5} m_{i,j} w_{i,j}},$$

where $\kappa = 1$ indicates an ideal agreement and $\kappa = 0$ indicates an agreement which could be completely justified by chance. The Cohen's kappa statistics for each ranker pair are shown in figure 5.3.

Percentage agreement Another statistic calculated for the obtained ratings involved treating the Likert scale as an interval scale (*i.e.* assuming that besides the implicit rank order between the response categories, the intervals between them are also equal).[2] To calculate the interval-based agreement between two judges i and j, their picture ratings are assembled into 103-dimensional vectors $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$. Then the cumulative disagreement between these rating vectors can be calculated as $||\boldsymbol{r}_i - \boldsymbol{r}_j||_1$. Since the maximum cumulative disagreement $||\boldsymbol{r}_{\max}||_1$ is equal to $103 \times (5-1) = 412$, the percentage agreement between judges i and j can be calculated as

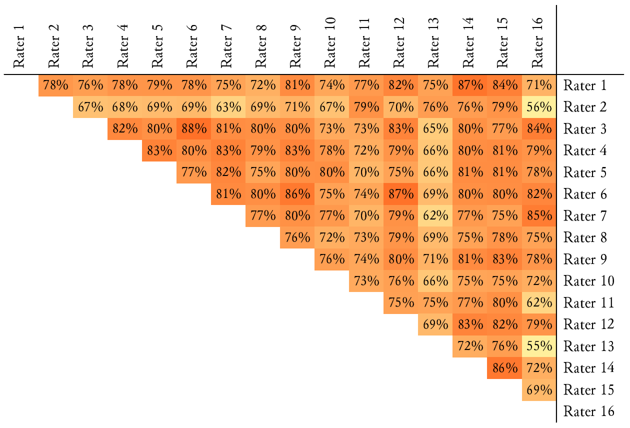
$$100 \times \left(1 - \frac{||\boldsymbol{r}_i - \boldsymbol{r}_j||_1}{||\boldsymbol{r}_{\max}||_1}\right) = 100 \times \left(1 - \frac{||\boldsymbol{r}_i - \boldsymbol{r}_j||_1}{412}\right).$$

These percentage agreements between each pair of judges are reported in figure 5.4.



Rater 1	Rater 2	Rater 3	Rater 4	Rater 5	Rater 6	Rater 7	Rater 8	Rater 9	Rater 10	Rater 11	Rater 12	Rater 13	Rater 14	Rater 15	Rater 16	
	0.45	0.47	0.28	0.29	0.56	0.23	0.22	0.49	0.21	0.43	0.64	0.37	0.63	0.55	0.34	Rater 1
		0.38	0.26	0.22	0.40	0.18	0.33	0.34	0.22	0.56	0.35	0.33	0.38	0.43	0.14	Rater 2
			0.46	0.39	0.77	0.39	0.54	0.52	0.23	0.48	0.62	0.30	0.56	0.47	0.55	Rater 3
				0.25	0.42	0.27	0.37	0.42	0.20	0.28	0.35	0.16	0.34	0.39	0.24	Rater 4
					0.39	0.25	0.28	0.36	0.30	0.22	0.25	0.13	0.32	0.32	0.21	Rater 5
						0.45	0.50	0.69	0.25	0.48	0.75	0.37	0.60	0.52	0.54	Rater 6
							0.32	0.37	0.19	0.21	0.40	0.18	0.22	0.25	0.43	Rater 7
								0.39	0.22	0.35	0.48	0.29	0.34	0.45	0.30	Rater 8
									0.21	0.38	0.54	0.35	0.52	0.51	0.41	Rater 9
										0.33	0.31	0.18	0.17	0.18	0.13	Rater 10
											0.47	0.42	0.42	0.49	0.20	Rater 11
												0.35	0.65	0.53	0.46	Rater 12
													0.26	0.37	0.19	Rater 13
														0.54	0.31	Rater 14
															0.28	Rater 15
																Rater 16

Figure 5.3: The inter-rater agreement coefficients measured using Cohen's (1968) weighted kappa method (with quadratic weights). The agreement's strength is indicated by the blue colour's intensity (where an agreement arising totally by chance would be indicated by $\kappa = 0$ and would be coloured in white).



Rater 1	Rater 2	Rater 3	Rater 4	Rater 5	Rater 6	Rater 7	Rater 8	Rater 9	Rater 10	Rater 11	Rater 12	Rater 13	Rater 14	Rater 15	Rater 16	
	78%	76%	78%	79%	78%	75%	72%	81%	74%	77%	82%	75%	87%	84%	71%	Rater 1
		67%	68%	69%	69%	63%	69%	71%	67%	79%	70%	76%	76%	79%	56%	Rater 2
			82%	80%	88%	81%	80%	80%	73%	73%	83%	65%	80%	77%	84%	Rater 3
				83%	80%	83%	79%	83%	78%	72%	79%	66%	80%	81%	79%	Rater 4
					77%	82%	75%	80%	80%	70%	75%	66%	81%	81%	78%	Rater 5
						81%	80%	86%	75%	74%	87%	69%	80%	80%	82%	Rater 6
							77%	80%	77%	70%	79%	62%	77%	75%	85%	Rater 7
								76%	72%	73%	79%	69%	75%	78%	75%	Rater 8
									76%	74%	80%	71%	81%	83%	78%	Rater 9
										73%	76%	66%	75%	75%	72%	Rater 10
											75%	75%	77%	80%	62%	Rater 11
												69%	83%	82%	79%	Rater 12
													72%	76%	55%	Rater 13
														86%	72%	Rater 14
															69%	Rater 15
																Rater 16

Figure 5.4: The inter-rater agreement percentages between each pair of judges (raters), calculated using the complements of the cumulative disagreement and a maximum possible disagreement ratios. An ideal agreement would be indicated by 100% (coloured in dark orange), and a complete disagreement would be indicated by 0% (coloured in white).

Since both metrics indicate the reliability of collected ratings, these ratings are compared to the ones obtained by autonomous robot photographers of Byers et al. (2003) and Ahn et al. (2006), as described below.

[2]Note that both intervalist/ordinalist views of the Likert scales and the applicability of parametric/non-parametric statistical analysis techniques are somewhat controversial (*e.g.* see the scientific debates in research journals by Jamieson (2004), Carifio and Perla (2008) and Norman (2010)). The validity of each view-point largely depends on the underlying data that is being categorized by these scales, type of postprocessing performed on the collected responses (*e.g.* averaged or summed responses are easier to justify as being interval-like), overall size of the surveys and so on. The evaluation performed in this thesis considers the obtained results from both perspectives.

5.1.1.3 Comparison to ratings obtained by other robot photographers

Most of the robot photographer development approaches described in the literature and summmarized in chapter 2 either did not provide thorough quantitative evaluations of the obtained photographs (including Campbell and Pillai (2005), Kim et al. (2010) and Shirakyan et al. (2012)) or the approaches were not directly comparable to the one used in this project (*e.g.* the robot photographer developed by Gadde and Karlapalem (2011) was stationary and took pictures only of static scenes).

For this reason, the results obtained by Luke are compared to the results obtained in the approaches described by Byers et al. (2003) and Ahn et al. (2006), who provided the quantitative evaluations of the pictures that their robots took on the Likert scales.

To that end, the comparison of photograph proportions in each of the rating categories for each robot photographer approach are given in table 5.2 and figure 5.5, and the statistical summary of the results (using both-parametric and non-parametric statistics) is provided in table 5.3.

Authors/Ratings	Very bad (1)	Bad (2)	Neutral (3)	Good (4)	Very good (5)
Byers et al. (2003)	18.0%	25.0%	28.0%	20.0%	9.0%
Ahn et al. (2006)†	6.7%	23.5%	32.7%	26.0%	11.1%
Zabarauskas (2013)‡	4.7%	14.6%	25.5%	33.5%	21.7%

† Ahn et al. used the following Likert scale: "*Very poor*", "*Poor*", "*Normal*", "*Nice*", "*Very nice*".
‡ Work completed and presented in this dissertation.

Table 5.2: Proportion of pictures in each of the rating categories in three different robot photographer approaches.

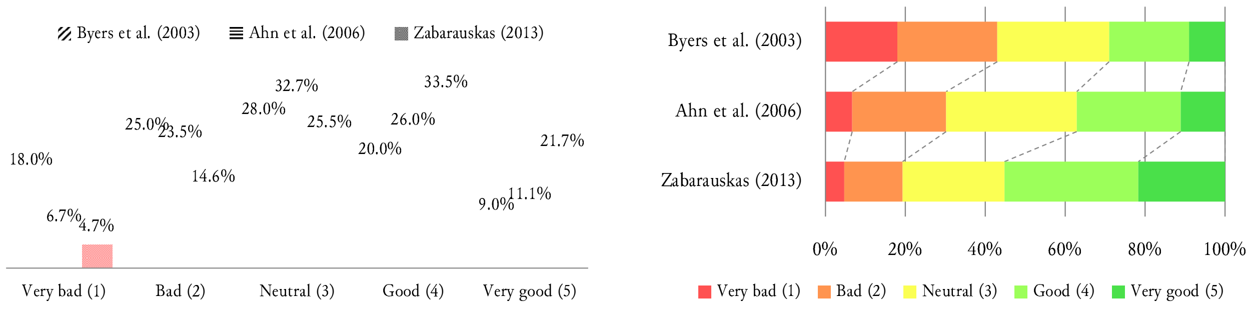


Figure 5.5: Visual summary of the proportion of pictures in each of the rating categories in three different robot photographer approaches.

As shown in table 5.2 and figure 5.5, more than half of all pictures taken by Luke were evaluated by humans as being either "good" or "very good", and more than 80% were evaluated as "neutral" or better, significantly outperforming Byers et al.'s and Ahn et al.'s robots.

To verify the statistical significance of the obtained results, both parametric and non-parametric methods are employed to reject the hypothesis that the real rating means (and therefore the underlying qualities of the pictures taken by each robot) are equivalent, and the differences observed in the individual rating sets (as summarized in table 5.2/figure 5.5) arise purely by chance.

First, a parametric test based on one-way analysis of variance (ANOVA, Fisher (1924)) is performed on all three rating sets (*viz.* the ones obtained by Byers et al. (2003), Ahn et al. (2006) and Zabarauskas (2013)) simultaneously.

This approach is slightly controversial because some researchers consider Likert scales only as ordinal data. Such data should therefore be analysed with non-parametric methods, while ANOVA assumes normal distribution of

Authors	Central tendency			Variability	
	Mean	Median	Mode	Standard deviation	Inter-quartile range
Byers et al. (2003)†	2.77	Good (3)	Good (3)	1.22	2
Ahn et al. (2006)	3.11	Good (3)	Good (3)	1.09	2
Zabarauskas (2013)	3.53	Very good (4)	Very good (4)	1.12	1

† Since Byers et al. mention that around 2,000 photographs were evaluated (without specifying the exact number), the comparisons in the following sections assume an exact rating count of 2,000.

Table 5.3: Comparison of parametric and non-parametric statistics of the pictures taken by different robot photographers.

responses and thus it is a parametric technique. However, more recent literature argues for the applicability of such techniques when analysing Likert scale responses. For example, Carifio and Perla (2008) state that "*it is perfectly appropriate to summarise the ratings generated from Likert scales using means and standard deviations, and it is perfectly appropriate to use parametric techniques like Analysis of Variance to analyse Likert scales.*"

Another frequent criticism of ANOVA's use for Likert scale analysis is that while the original ANOVA test assumes the normality of the response variable distribution, the Likert scale responses tend to be skewed. However, as noted by Kirk (1995) (and others), ANOVA is relatively robust *w.r.t.* violations of this assumption. This is also suggested by Norman (2010), who counters these two common criticisms by stating that "*parametric statistics can be used with Likert data, [...] and with non-normal assumptions, with no fear of "coming to the wrong conclusion.*"

In the light of these arguments, the details of the ANOVA method (within the context of multiple sets of picture ratings) are described below.

A one-way analysis of variance (ANOVA) In ANOVA test, the real quality of the pictures taken by each robot is modelled by an underlying random variable, where individual samples of this underlying variable (with an associated sampling error) are obtained by rating pictures that the robot took. (To that end, it is assumed that both the judges and the ratings are independent from each other, *i.e.* the rating of a given picture by one judge does not affect another judge's rating of the same picture in any way, and, similarly, that for a given judge, rating one picture does not affect the rating for another picture.) Then, the variance between the mean ratings of each set is compared to the variance between individual ratings. If the ratio of these variances is high enough (*i.e.* unlikely to occur by chance), then ANOVA test considers that photo ratings must have come from distributions which have sufficiently different mean ratings. This implies that if ANOVA's null hypothesis of equal underlying population means is rejected, then there is statistical difference between the real picture quality for each of the robots.

In particular, ANOVA test for photo ratings works as follows. Let $x_{i,j}$ be the observed rating for photo j in the rating set i (where each set contains n_i ratings and there are S sets in total). Furthermore, let the unbiased estimate of the mean of rating set i be $\bar{x}_i = \frac{\sum_j x_{i,j}}{n_i}$, and the unbiased estimate of rating set i's variance be $s_i^2 = \frac{\sum_j (x_{i,j} - \bar{x}_i)^2}{n_i - 1}$. Similarly, let $\bar{x} = \frac{\sum_{i,j} x_{i,j}}{N}$ be the unbiased estimate for the mean of combined rating sets, obtained by pooling together ratings from all sets (where $N = \sum_i n_i$). Then the "sum of squares" measure *between* the rating sets can be calculated as

$$SS_{\text{between}} = \sum_i n_i(\bar{x}_i - \bar{x})^2,$$

and the "sum of squares" measure *within* the rating sets can be computed as

$$SS_{\text{within}} = \sum_i s_i^2(n_i - 1).$$

The F-score for the ANOVA test can be calculated as a ratio of the mean squares between and within the rating sets (where a mean square is defined as the sum of squares divided by its degrees of freedom), namely:

$$F = \frac{SS_{\text{between}}/df_{\text{between}}}{SS_{\text{within}}/df_{\text{within}}} = \frac{(N-S)\sum_i n_i(\bar{x}_i - \bar{x})^2}{(S-1)\sum_i s_i^2(n_i-1)}, \tag{5.1}$$

where $df_{\text{between}} = S - 1$ and $df_{\text{within}} = N - S$.

To be able to reject the null hypothesis (*viz.* the equality of mean ratings for all S rating sets), F-score has to exceed the critical value threshold $F^{\text{crit}}_{df_{\text{b.}},df_{\text{w.}}}(\alpha) = F^{\text{crit}}_{S-1,N-S}(\alpha)$ for a desired significance level α, where $F^{\text{crit}}_{S-1,N-S}$ is the inverse of a cumulative distribution function (CDF) of F-distribution[3] with $S-1$ and $N-S$ degrees of freedom.

In order for ANOVA test to be applicable (besides the independence assumption described above), the responses ideally should be normally distributed. While it is not obvious whether the rating sets by Zabarauskas (2013), Ahn et al. (2006) and Byers et al. (2003) satisfy this condition, they are also not extremely skewed (the kurtosis of each set respectively is -0.614, -0.71 and -0.92; it can also be seen by visual inspection of the left half of figure 5.5). Due to the fact that ANOVA test is not extremely sensitive to non-normality, it is still considered to be applicable.

However, a standard ANOVA test also relies on homoscedasticity[4] of the rating sets. Since the variances of the rating sets collected by Zabarauskas (2013), Ahn et al. (2006) and Byers et al. (2003) are 1.261, 1.198 and 1.478 respectively, it is unlikely that the homoscedasticity condition holds in this situation. To mitigate this problem, an ANOVA-based $F^\star$ test by Brown and Forsythe (1974a) [5] is used, which is designed for sample sets with non-equal variance (and is less sensitive to non-normality than the regular ANOVA test). This test is described in more detail below.

Brown-Forsythe $F^\star$ test To test whether all three rating sets have equal means, given that they have unequal variances, the Brown and Forsythe (1974a) test redefines the F-score from the equation 5.1 as

$$F^\star = \frac{SS_{\text{between}}}{\sum_i s_i^2\left(1-\frac{n_i}{N}\right)} = \frac{\sum_i n_i(\bar{x}_i - \bar{x})^2}{\sum_i s_i^2\left(1-\frac{n_i}{N}\right)}, \tag{5.2}$$

where the critical value threshold $F^{\text{crit}}_{S-1,df_{\text{within}}}(\alpha)$ for a desired significance level α is obtained from the same CDF inverse of F-distribution. Here the degrees of freedom within the rating sets df_{within} are derived from Satterthwaite's (1941) equation as

$$df_{\text{within}} = \left(\sum_i \frac{c_i^2}{n_i - 1}\right)^{-1} \quad \text{where} \quad c_i = \frac{\left(1-\frac{n_i}{N}\right)s_i^2}{\sum_j\left(1-\frac{n_j}{N}\right)s_j^2}.$$

Running the $F^\star$ test on the rating sets obtained by Byers et al. (2003), Ahn et al. (2006) and Zabarauskas (2013) produces an $F^\star$ score of 199.84. For the p-value of 0.0001, the critical value $F^{\text{crit}}_{2,4212.1}(0.9999) = 9.23$. Since $F^\star \gg F^{\text{crit}}_{2,4212.1}(0.9999)$, the probability that the null hypothesis of equal means for these rating sets holds is smaller than 0.0001, and hence it is rejected under this level of significance.

[3]Also known as Fisher-Snedecor distribution.

[4]Also known as the homogeneity/equality of variances.

[5]$F^\star$ test is used for the equality of *means* and should not be confused with a different Brown and Forsythe (1974b) test for the equality of *variances*.

Due to some degree of controversiality regarding the use of ANOVA with data obtained using Likert scales, the same hypothesis is tested using a non-parametric version of one-way analysis of variance by Kruskal and Wallis (1952). This test is described below in more detail.

Kruskal-Wallis one-way analysis of variance In contrast to the original ANOVA method (as described above), the test proposed by Kruskal and Wallis (1952) does not assume an underlying normal distribution or the homogeneity of variances. It works in the following way.

First, all N ratings from all rating sets are pooled together and ranked, yielding ranks $r_{i,j}$ where j is a rating from the rating set i. If k ratings are tied for the ranks $n+1, ..., n+k$, they are assigned the average rank $\frac{1}{k}\sum_{i=1}^{k}(n+i) = n + \frac{k+1}{2}$.

Let $\bar{r}_i = \frac{\sum_j r_{i,j}}{n_i}$ be the average rank in the rating set i and $\bar{r} = \frac{\sum_{i,j} r_{i,j}}{N} = \frac{N+1}{2}$ be the average rank of all ratings in the pooled rating set. Then the equation (5.1) is re-defined to use the ranks of the ratings (instead of the actual ratings) as

$$K = (N-1)\frac{\sum_i n_i(\bar{r}_i - \bar{r})^2}{\sum_{i,j}(r_{i,j} - \bar{r})^2}. \tag{5.3}$$

Since K approximately follows the chi-squared distribution with $S-1$ degrees of freedom (*i.e.* $K \sim \chi^2_{S-1}$, where S is the number of rating sets), the obtained K-score can be used to reject the null hypothesis (*viz.* the equality of mean ranks of ratings for all S rating sets) if it exceeds the critical value $K^{\text{crit}}_{S-1}(\alpha)$ for a desired significance level α. Here K^{crit}_{S-1} is the inverse CDF for χ^2_{S-1} distribution.

Performing this test on Byers et al.'s, Ahn et al.'s and Zabarauskas' rating sets results in a K score of 353.12. For the p-value of 0.0001, the critical value $K^{\text{crit}}_2(0.9999) = 18.42$. Since $K \gg K^{\text{crit}}_2(0.9999)$, the null hypothesis of equal mean ranks of ratings is also rejected using this non-parametric method (as summarized in table 5.4).

Statistical test	Test score	Critical score ($p = .0001$)	Null hypothesis	Decision
Parametric (Brown-Forsythe)	$F^\star = 199.8352$	$F^{\text{crit}} = 9.2305$	"Means of underlying rating populations are equal."	Rejected
Non-parametric (Kruskal-Wallis)	$K = 353.1212$	$K^{\text{crit}} = 18.4207$	"Mean ranks of underlying rating populations are equal."	Rejected

Table 5.4: Parametric and non-parametric statistical test results for the significance of the similarity between Byers et al. (2003), Ahn et al. (2006) and Zabarauskas (2013) rating sets.

Given these results, the following section examines each rating set pair individually to determine which of the robot photographer approaches produce statistically significant differences in picture quality.

5.1.1.4 Post-hoc testing for significant differences between rating set pairs

Since both parametric and non-parametric tests confirm statistically significant differences between the rating set means simultaneously (at $p = .0001$ confidence level), but they do not specify whether (and which) individual pairs of rating sets come from different underlying distributions, a number of pair-wise post-hoc tests are performed.

First of all, each pair of rating sets is compared using a version of parametric Student's t-test (Gosset, 1908). This test can be used to reject the null hypothesis that the underlying populations from which the two rating sets have been sampled actually have equal mean ratings, and the observed differences arise purely by chance. However, this test also assumes homoscedasticity, which (as discussed) above cannot be easily shown for the rating sets. While it is relatively insensitive to violations of this assumption when the sample sizes are equal, it is not very robust in presence of unequal variances and unequal sample sizes. Welch (1947) proposes a version of Student's t-test, which is insensitive to unequal variances even for different-size samples. This version of the t-test is described in more detail below.

Welch's t-test To check whether unequal means observed between individual pairs of rating sets could have arisen by chance, the modified t-test (as described by Welch (1947)) is employed. Besides the assumption for rating independence within/between rating sets, this test also requires that the means of the rating sets would be distributed normally. Since each set contains more than a $1,000$ pictures, it is assumed that this requirement is satisfied by the central limit theorem. Furthermore, as described above, this test is robust to unequal variances and differing sample sizes. It works in the following way.

As earlier, let $\bar{x}_i = \frac{\sum_j x_{i,j}}{n_i}$ and $s_i^2 = \frac{\sum_j (x_{i,j} - \bar{x}_i)^2}{n_i - 1}$ be the unbiased estimates of rating set i mean/variance respectively, where n_i is the size of rating set i. Then the t-score, defined as

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{(s_1^2/n_1) + (s_2^2/n_2)}} \tag{5.4}$$

is distributed following the Student's t distribution with

$$\nu = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\left(\frac{s_1^2}{n_1}\right)/(n_1 - 1) + \left(\frac{s_2^2}{n_2}\right)/(n_2 - 1)}. \tag{5.5}$$

degrees of freedom. (Here ν is approximated using the Welch-Satterthwaite equation, since it arises from a linear combination of individual rating set variances.)

After the t-scores are obtained for each of the rating set pairs, they are individually compared to the critical value $t_\nu^{\text{crit}}(\alpha)$ at the significance level α, where t_ν^{crit} is calculated from the inverse CDF F_ν^{-1} of the Student's t-distribution.

Since the null hypothesis assumes that the real means of the underlying rating populations are equal, the rejection of this hypothesis could come from either "tail" of the symmetrical t-probability distribution, hence the critical value for a specified statistical significance level α is calculated from

$$\begin{gathered}
\mathrm{Pr}_\nu(|t| \leq t_\nu^{\text{crit}}) = \alpha \Leftrightarrow \\
1 - 2\mathrm{Pr}_\nu(|t| > t_\nu^{\text{crit}}) = \alpha \Leftrightarrow \\
t_\nu^{\text{crit}} = F_\nu^{-1}\left(\frac{\alpha + 1}{2}\right).
\end{gathered}$$

To avoid inflating the Type I error[6] by performing repeated null hypotheses tests, the Bonferroni correction (Dunn, 1961) is used. In particular, since three tests are required to check each pair of the rating sets for the equality of their

[6]Rejecting the null hypothesis when it in fact holds.

means, the p-value of 0.0001 (as used above) is discounted by a factor of three, *viz.* $p = 0.0001/3 = 0.0000333$ and thus $\alpha = 99.9967\%$. For this level of statistical significance, Welch's t-test rejects the null hypotheses of mean equality for each of the rating set pairs, as summarized in table 5.5.

Since Welch's t-test interprets the Likert scales as interval data (which is somewhat controversial, as discussed above), the same post-hoc tests are repeated using a non-parametric analogue of t-test as proposed by Mann and Whitney (1947).

Mann-Whitney U test[7] To compare the rating set pairs using the non-parametric techniques, the test described by Mann and Whitney (1947) is used. In contrast to Welch's t-test, this test does not assume interval data (*i.e.* it can be straight away applied to ordinal data, like Likert scales).

Mann-Whitney's U test can be used to refute similar null hypothesis as the Welch's t-test (*viz.* the assumption that the two rating sets containing n_1 and n_2 ratings come from the same distribution) in the following way.

First of all, $n_1 + n_2$ ratings from both rating sets are ranked collectively, yielding ranks $r_{i,j}$ for each rating $j \in \{1, ..., n_i\}$ from each rating set $i \in \{1, 2\}$. Tied ranks are assigned the average rank, like in Kruskal-Wallis method above. Let $r_i = \sum_j r_{i,j}$ be sum of all ranks in rating set i. Then the U-score can be calculated as

$$U = \min \left\{ n_1 n_2 + \frac{n_1(n_1+1)}{2} - r_1, \;\; n_1 n_2 + \frac{n_2(n_2+1)}{2} - r_2 \right\}.$$

Given that for the large n_1, n_2 values $U \sim \mathcal{N}(\mu, \sigma)$, with $\mu = \frac{n_1 n_2}{2}$, $\sigma = \sqrt{\frac{n_1 n_2 (n_1+n_2+1)}{12}}$, the critical value $U^{\text{crit}}(\alpha)$ for null hypothesis rejection with the significance level α can be found using the inverse CDF of the normal distribution with parameters μ and σ, *i.e.* $U^{\text{crit}}(\alpha) = \Phi^{-1}_{\mu,\sigma}(\alpha)$. Under the same Bonferroni correction, p-value of $0.0001/3$ is chosen for each test of a rating set pair. For this level of statistical significance, Mann-Whitney U test also rejects the "equal distribution" null hypotheses for each of rating set pairs, as summarized in table 5.5.

Statistical test	Rating set pair	Test score	Critical score ($p = .000033$)	Null hypothesis	Decision
Parametric (Welch's t-test)	Zabarauskas, Ahn et al.	$t = 9.4661$	$t^{\text{crit}} = 4.1578$	"Means of underlying rating populations are equal."	Rejected
	Zabarauskas, Byers et al.	$t = 19.543$	$t^{\text{crit}} = 4.1547$		Rejected
	Ahn et al., Byers et al.	$t = 7.8988$	$t^{\text{crit}} = 4.1576$		Rejected
Non-parametric (Mann-Whitney U-test)	Zabarauskas, Ahn et al.	$U = 675343$	$U^{\text{crit}} = 778807.65$	"Underlying rating populations have similar distribution."	Rejected
	Zabarauskas, Byers et al.	$U = 1079750$	$U^{\text{crit}} = 1521749.9$		Rejected
	Ahn et al., Byers et al.	$U = 872000$	$U^{\text{crit}} = 948443.07$		Rejected

Table 5.5: Parametric and non-parametric statistical test results for the significance of the pairwise similarity between Byers et al. (2003), Ahn et al. (2006) and Zabarauskas (2013) rating sets.

[7]Also known as Wilcoxon rank-sum test.

5.2 Summary

In this chapter, Luke's deployment at an open-day event in the Department of Computer Science, University of Oxford was described. During the three and a half hours of active photo shooting, the robot took 103 pictures, *i.e.* roughly one picture every two minutes. For evaluation, each of these pictures were rated by sixteen judges unrelated to the project. More than half of the pictures were rated as being "good (4)" or "very good (5)" on a five-point Likert scale, markedly exceeding the results reported by Byers et al. (2003) and Ahn et al. (2006). To confirm the statistical significance of these findings, both parametric (one-way ANOVA and pairwise Welch's t-tests) and non-parametric measures (Kruskal-Wallis and pair-wise Mann-Whitney U-tests) were taken. At $p = .0001$ significance level, both types of tests indicated statistically significant differences between the obtained results.

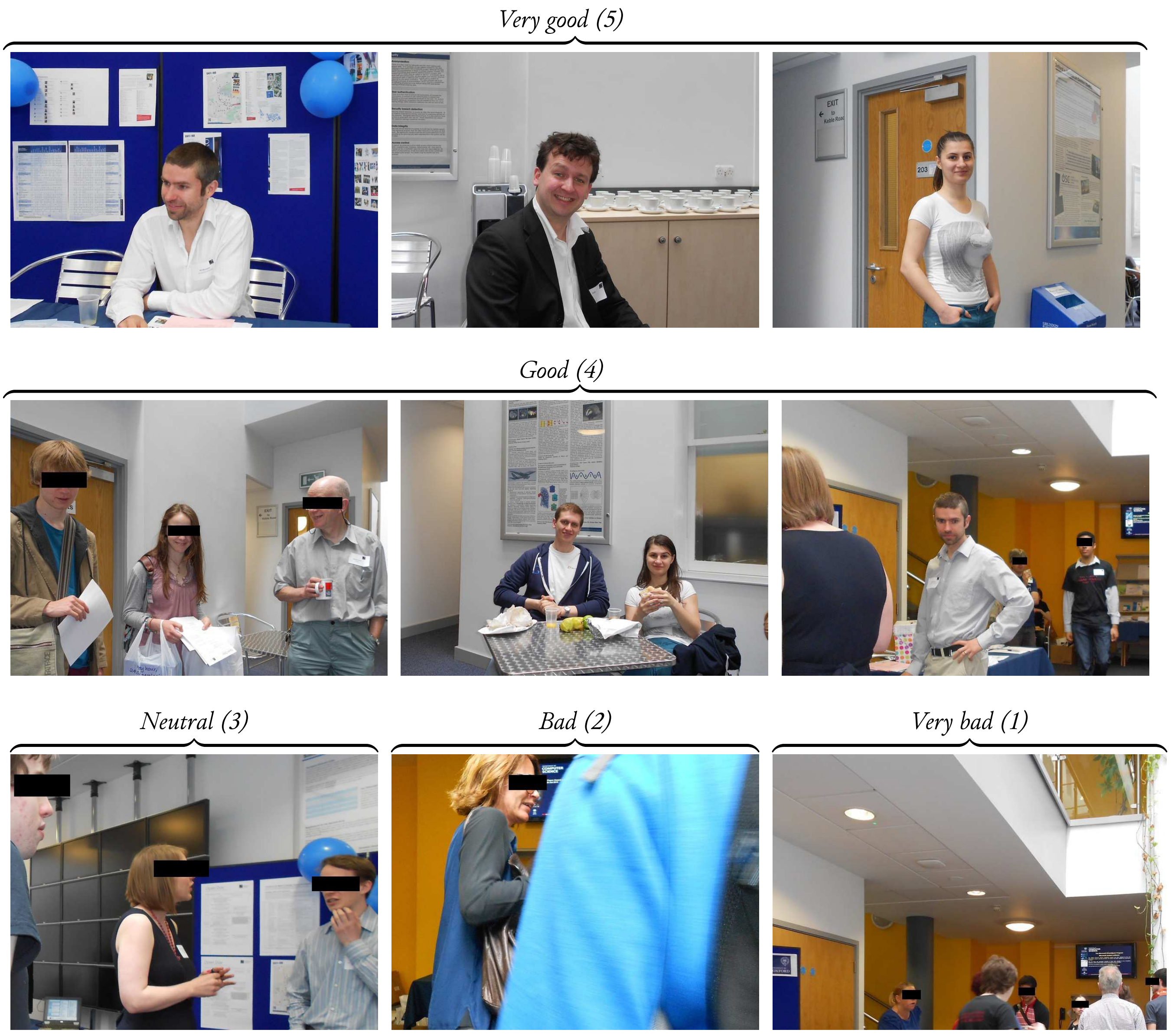


Figure 5.6: Luke's photographs with the largest rating count in each of the categories.

Chapter 6

Conclusions and Proposals for Further Research

This final chapter summarizes the work completed in this dissertation, sums up the contributions of this thesis to the field of autonomous robot photography, and outlines the potential directions for future research.

6.1 Summary of completed work

As discussed in Chapter 1, autonomous robot photographers serve as excellent low-cost robotics research platforms, encompassing difficult multidisciplinary challenges. However, all such systems described in the scientific literature within 2003–2013 (since the earliest robot photographer, to the present day, as summarized in Chapter 2) rely on RGB cameras for their vision and laser range/infrared/ultrasound sensors for obstacle detection and avoidance. Each of these devices have their own disadvantages *w.r.t.* low-cost and ubiquitous RGB-D sensors (as summarized in Chapter 3), and a number of autonomous robot systems, ranging from robotic wheelchairs (Wu et al., 2013) to home-service "robot butlers" (Stückler and Steffens, 2011), have already adopted RGB-D sensors for their vision systems. It was these recent developments of RGB-D sensors and their applications to different autonomous robots that inspired the investigation into how such sensors could be used to improve the state-of-the-art performance of autonomous robot photographers.

This project proposed RGB data-based solutions for both of the main challenges of robot photographers, *viz.* human subject detection/tracking and obstacle detection/avoidance. To make an informed decision, an extensive survey of both approaches based on RGB-D (separate and combined) data has been performed and presented in Chapter 3. The following solutions have been proposed based on their applicability to RGB-D data, their computational efficiency, keeping in mind the processing power and energy constraints imposed by mobile robotics, and their feasibility for implementation during the project's timeframe.

6.1.1 Contributions to the field of autonomous robot photography

Human subject detection/tracking For human subject detection in depth data, this thesis proposed an extension of Garstka and Peters' (2011) knowledge-based head localization method. This extension enabled simultaneous multiple head detection in depth data, as described in section 3.3.1.2.

To increase the accuracy of this approach, this thesis also proposed a novel method for detected "candidate" head verification using RGB data. In particular, this method employed a Viola and Jones (2001) face detector cascade to gather a small set of skin/background training examples for every new environment that the robot was deployed in. It then used these examples to train the discriminative logistic regression classifier with RBF kernel on-the-spot, as discussed in section 3.3.1.3.

This thesis also presented and incorporated an already existing skin-classification approach for depth-based head detection verification using RGB data. In this approach a histogram-based Bayesian skin classifier was trained on a large-scale supervised data set, containing almost a billion of skin/non-skin training examples, following the description of Jones and Rehg (2002).

Lastly, to increase the efficiency of the proposed RGB-D head detection approach and to reduce the impact of noise in the sensor's RGB-D stream, this thesis also proposed a new depth-based extension of the continuously-adaptive mean-shift tracker (Bradski, 1998), yielding an efficient method of tracking detected human heads in depth data over a sequence of input frames (see section 3.3.1.4).

Obstacle detection To enable the robot photographer's "random wandering" mode, Boucher's (2012) depth-based obstacle detection and avoidance method was chosen. This method was combined with Peasley and Birchfield's (2013) heuristic for close-range obstacle avoidance. With only minor modifications (like the incorporation of the accelerometer's data from the Kinect sensor, and feedback from the bumpers on the robot's base) this combined method proved to be robust and efficient enough to be used in an unstructured environment, under a flat ground assumption.

6.1.2 Implementation and evaluation of the proposed methods

The methods described above were implemented within an open-source Robot Operating System framework (Quigley et al., 2009), based on Brooks' (1986) behaviour-based architectural design (as thoroughly described in Chapter 4), achieving sufficient performance for real-time applications on modest configuration on-board netbooks/laptops/PCs[1].

To test this software in real-world situations, a physical robot has been built using an open-source hardware platform (available off-the-shelf, or as a construction kit), a simple point-and-shoot photographic camera and a low-cost RGB-D Microsoft Kinect sensor. Only affordable and widely available (or interchangeable) parts were used in robot's construction to promote reproducibility of the results and facilitate research on autonomous mobile robots both within and outside academia.

The developed autonomous photographer robot "Luke" has been deployed in an unstructured open-day event, as described in Chapter 5. During this event, Luke took more than a hundred pictures, shooting a new photograph roughly every two minutes of its operation. All taken pictures were evaluated by sixteen judges (unrelated to the project), yielding a total of 1,648 ratings which were determined to be sufficiently reliable by inter-rater agreement measurements.

More than half of Luke's pictures were rated by judges as being "good" or "very good" on a five-point Likert scale, markedly exceeding the results reported by Byers et al. (2003) and Ahn et al. (2006). The statistical significance of these results has been confirmed by both parametric (one-way ANOVA and pairwise Welch's t-tests) and non-parametric tests (Kruskal-Wallis and pair-wise Whittney U-tests) at $p = .0001$ significance level.

Based on these results, the project is considered to have achieved all proposed aims and satisfied all success criteria set out in section 1.4.

The following final section of this thesis describes some limitations of the current approach, and proposes directions for further autonomous robotic photographer research.

6.2 Current approach limitations and directions for future research

As discussed earlier, the proposed methods for human subject detection/tracking and obstacle detection/avoidance were chosen due to their computational efficiency (enforced by the simplicity of the on-boards netbook) and feasibility to be implemented within the project's timeframe. While the proposed RGB-D data based methods

[1]The source code of the implemented robot photographer will be available after the submission of this thesis at `http://zabarauskas.com/robot-photographer`.

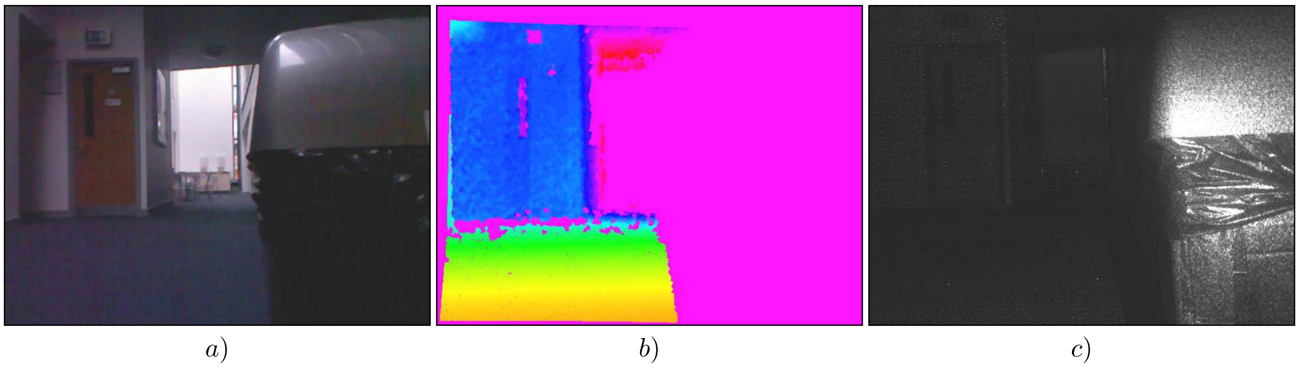


Figure 6.1: Potential application of Kinect's infrared camera/projector pair for close obstacle detection. Image a) shows the RGB input from Kinect's colour camera, image b) shows the corresponding depth image, with missing depth data coloured in purple, and image c) shows the Kinect's infrared camera image of the same scene. Notice that while the depth data missing due to nearby obstacles/out of range parts of the scene is indistinguishable in the depth image, the projected infrared stuctured light pattern on the nearby obstacle is clearly visible in image c) (*c.f.* with the intensity of this pattern when projected on the far wall).

were sufficiently effective to advance the current state-of-the-art in autonomous robotic photography, additional improvements to these approaches could further enhance the performance of robot photographers.

In particular, improving human subject detection and tracking should directly lead to the enhancements in the quality of taken pictures. This is because the photo composition approach described by Dixon et al. (2003) directly depends on the accuracy of human localization in the photographic camera's image plane; indeed, the majority of badly rated pictures in this thesis were aesthetically unpleasant because of incorrectly located human subject heads in RGB-D image.

Regarding the obstacle detection and avoidance, most of the limitations in the described approach stem from the hardware constraints of the Kinect sensor. For example, the visible depth range of the sensor is between ~0.7 m–4.0 m, which implies that any obstacles closer than 70 centimetres to the sensor cannot be detected. By attaching Kinect towards the back of the robot's base, the "blind" area in front of the robot can be reduced to around 40 centimetres.

While the presence of large obstacles in this area can be approximated from the loss of depth readings, this can lead to false positive obstacle detections (which is not too severe for "random wandering" mode). However, small obstacles cannot be detected even using these heuristics, and thus other sources of information, like bumper sensors, have to be used.

Two potential approaches can be taken to mitigate this problem: the first approach would be to attach another RGB-D sensor in front of the robot, facing downwards from around 70 centimetre height. This would not cause any IR interference with the sensor used for human detection/tracking, and would eliminate any blind spots in front of the robot. Of course, other inexpensive sensors (*e.g.* ultrasound) could also be used for this task.

Another, more interesting solution for obstacle detection in Kinect's blind zone is to exploit the embedded IR projection/capture technology that Kinect sensor uses to infer depths in the scene.

In particular, data loss in the depth image can occur due to objects being too far or too close to the sensor; these cases are indistinguishable in the retrieved depth data. However, the structured light pattern emitted by Kinect's IR projector is clearly visible in the sensor's IR camera image on the obstacles positioned closer than 70 centimetres to the sensor, and, conversely, this light pattern has very low intensity when projected on objects further than the far-range limit of the sensor (*i.e.* further than 4 metres), as shown in image 6.1. Using image processing

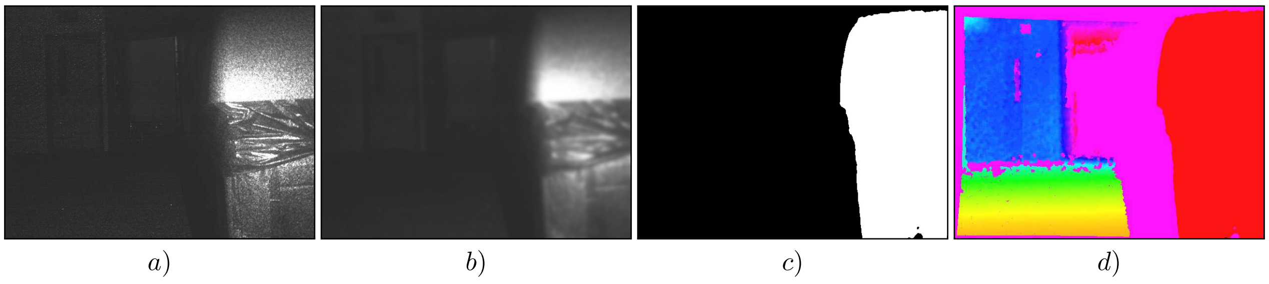


Figure 6.2: Potential approach for infrared data combination with the depth data for obstacle avoidance. Image *a*) shows the raw infrared input from Kinect's IR camera, image *b*) shows this image after convolution with a Gaussian kernel, image *c*) shows the result obtained after intensity thresholding and image *d*) shows the combined depth and processed IR images for obstacle detection.

and/or machine learning techniques (with the basic example illustrated in figure 6.2), this property could be used to discriminate between these two cases.

Another significant hardware constraint of Kinect device is its limited vertical field-of-view (∼45°). When the same device is used both for obstacle and human subject detection, it imposes a difficult trade-off between the closest distance at which both the standing humans and the objects lying on the floor plane can be detected, as illustrated in figure 6.3. Possible solutions to this problem include mounting the Kinect sensor sideways (resulting in a vertical ∼58° FOV, but limiting the number of people visible horizontally at the same time), or using an additional sensor for obstacle detection, as described above.

Some further improvements of robot photographers could include proactive navigation planning. In particular, the robot could attempt to actively predict the locations in the environment which could yield good quality pictures based on last known positions of humans and try to navigate there. Similarly, landmark-tracking based localization systems could be integrated into the architecture to ensure that the robot does not wander outside of the designed photography area (basic implementations of both of these capabilities were described by Byers et al. (2003)).

Finally, a number of improvements to robot photographers could be made in human-robot interaction (HRI) area. As described by Smart et al. (2003), an event photographer robot should be capable of bimodal interactions: in the first mode (as presented in this thesis), an event photographer robot should remain relatively "inconspicuous" in the environment and take candid pictures of the subjects. In the second mode, users could drive the interaction with the robot by asking and giving instructions to the robot as to what pictures they would like to be taken. As a

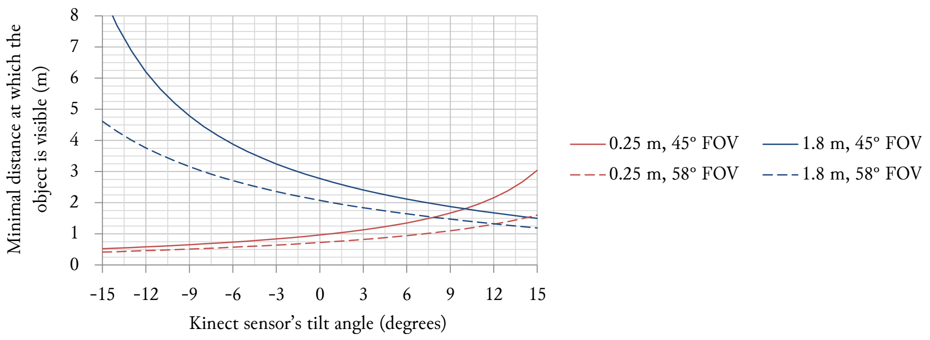


Figure 6.3: The shortest distances at which a 0.25 *cm* obstacle on the ground floor and the 1.8 *m* height human can be detected, as a function of Kinect's tilt angle (where the sensor parallel to the floor would have the tilt angle equal to zero, tilted towards the ceiling would have a positive tilt angle, and tilted towards the floor would have the negative tilt angle). These distances are presented for normal sensor's orientation as mounted on Luke (yielding around ∼45° vertical FOV) and the possible sideways orientation (yielding ∼58° vertical FOV).

simple example, if humans wave towards a robot randomly wandering about, the robot should direct its attention to them, approach the callers and take their picture. In more sophisticated versions of this reactive interaction mode, the robot should estimate the subject's intentions from more subtle cues, like the eye gaze, or speed/direction of movement.

In fact, HRI research area is of special importance to autonomous robot photographers. As noted by Byers et al. (2003), their applicability to this field stems from the fact that they perform a task which is i) understandable from the first glance, and ii) has a natural application in environments filled with humans. This simplicity of their functional purpose makes them approachable by people of various ages, education levels and general backgrounds, and invokes a variety of natural human-robot interactions. Due to this richness of naturally occurring interactions between them and humans, research into HRI aspects of autonomous robot photographers has potential to lead to completely novel HRI results. As the autonomous robots are slowly but surely making their way into our everyday lives (from vacuuming robots, to self-driving cars), this research is becoming more and more important.

Appendix A

A.1 Luke's State Externalization Messages

This section provides an excerpt from the state externalization table that Luke uses to generate vocal and text messages to communicate its state to humans, as discussed in section 4.2.3.3.

$S_{locomotion}$	$S_{head\ tracking}$	$S_{auton.\ phot.}$	$S_{framing}$	Message
PROCESSING_ BUMPER_EVENT	X	X	X	"Sorry!" "Apologies!" "My bad, sorry!"
¬PROCESSING_ BUMPER_EVENT	GATHERING_ FACE_HUE_DATA	X	X	"I'll learn the room's lighting conditions." "I'll look for the light sources in the room."
¬PROCESSING_ BUMPER_EVENT	¬GATHERING_ FACE_HUE_DATA	AVOIDING_ OBSTACLES	X	"I'll just look around for a few moments..." "I'll go and take some pictures from another angle..." "I'll take a small break for a few moments..."
¬PROCESSING_ BUMPER_EVENT	¬GATHERING_ FACE_HUE_DATA	TAKING_ PICTURE	X	"Smile, I'm taking a picture!" "Look at the camera, I'm taking a picture!" "Look up, I'm taking the picture!" "I'm taking the picture, say cheese!"
¬PROCESSING_ BUMPER_EVENT	¬GATHERING_ FACE_HUE_DATA	UPLOADING_ PICTURE	X	"Thank you! I'm saving the picture now..." "Thanks, the photograph looks perfect! I'm saving it..." "Thank you, that looks really good! Let me save it..."
¬PROCESSING_ BUMPER_EVENT	¬GATHERING_ FACE_HUE_DATA	FRAMING_ PICTURE	FRAME_ TOO_SMALL	"You're too far, I'm coming closer." "Hold on, I'll come a bit closer." "You're too far. Hold on a second." "I need to come a bit closer..."
¬PROCESSING_ BUMPER_EVENT	¬GATHERING_ FACE_HUE_DATA	FRAMING_ PICTURE	FRAME_OUT_ OF_BOUNDS	"The picture is off to one side... I'll try to center it!" "I'm trying to get a better photo composition. Hold on a second..." "Let me take a look from a different angle... Hold on." "Let me try to center the picture a bit..."
¬PROCESSING_ BUMPER_EVENT	¬GATHERING_ FACE_HUE_DATA	FRAMING_ PICTURE	NO_FRAME	"I cannot see anything interesting. I'll try from over there..." "I'm just looking for someone to photograph..."

Table A.1: An excerpt from Luke's state externalization phrase book (X is a "do not care" symbol).

A.2 Additional Examples of Discriminative Human/Face Detectors

This section describes a few additional examples of discriminative human/face detectors, based on the classifiers and human/face feature descriptors discussed in sections 3.1.1.3 and 3.1.2.1.

Image intensities with quadratic kernel SVMs A face detection approach that directly uses image intensities as input features has been proposed by Osuna et al. (1997). In their approach, 19×19 pixel size grayscale images are classified using a SVM with a quadratic kernel (with the applied elliptical mask to ignore window boundaries).

This SVM is trained in a supervised fashion, using datasets of preprocessed face images (with image intensities normalized using brightness correction and histogram equalization) and non-face images, bootstrapped from images of landscapes, trees, buildings and so on. After the training, the obtained 2,500 support vectors are used for sliding-window classification, over multiple scales of the input image.

PCA features with quadratic/Gaussian kernel SVMs Munder and Gavrila (2006) use the PCA features for training binary human/non-human classifiers in the following way.

First they extract principal components from a training set containing 4,800 human images of 18×36-pixel size. (The principal components with the associated largest eigenvalues are shown in figure A.1.) Then they use a supervised "human"/"non-human" training image set to train a set of quadratic/Gaussian kernel SVM discriminative classifiers using the image representations in the reduced-dimensionality space as the feature vectors.

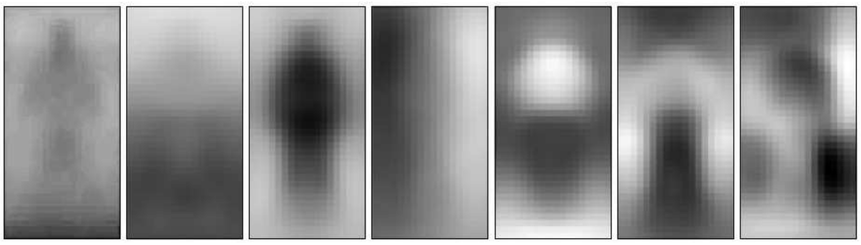


Figure A.1: Examples of principal components with the largest associated eigenvalues, obtained from a 4,800 human image training set (sorted in the order of decreasing eigenvalues). Adapted from Munder and Gavrila (2006).

Haar-like features with AdaBoost Viola et al. (2003) describe a human detector which uses the AdaBoost approach to select and construct "strong" classifiers from the Haar-like features, which perform the best on the "human"/"non-human" training set

Similarly to the approach described by Viola and Jones (2001), a cascade of these strong classifiers is assembled. However, in this approach the Haar-like features are evaluated not on the input images directly, but on the spatially shifted frame difference images (assuming a static camera position), to capture some of the temporal properties of human appearance.

LBP/HOG features with SVMs An example of the combined LBP and HOG feature use in human/face detection is presented by Wang et al. (2009). In their sliding-window approach, the presence/absence of humans in individual windows is verified using a combination of three linear SVMs (full body, upper body and lower body), which are trained based on LBP and HOG features.

In Wang et al.'s system, the occlusion likelihood is also modelled by the responses of the full-body detection SVM to individual blocks within the detector's window: if these blocks are within SVM's margin then the individual upper-/lower-body SVM classifiers are invoked to verify the presence of humans in the window.

Another example of a sliding-window human detector using HOG features is provided by Maji et al. (2008). In this approach, Maji et al. describe an efficient way to train a SVM with an intersection kernel ($K(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \min(h_a(i), h_b(i)$, where h_a and h_b are input histograms), using 1,360 HOG features.

HOD features with linear SVM Choi et al. (2013) proposes an improvement to Spinello and Arras's (2011) HOD-based human detection approach (fully described in section 3.1.2.1), in which a graph-segmentation algorithm is used to first segment the images into candidate regions based on the depth values and surface normals. After rejecting image segments which are unlikely to contain humans (based on geometric heuristics), the remaining regions are classified as human/non-human using a linear SVM with HOD descriptors.

The initial segmentation and region rejection steps allow Choi et al. to significantly improve the human detection speed while achieving similar accuracy in comparison to Spinello and Arras.

A.3 Obstacle Detection and Avoidance Source Code

This section provides a source code example of a ROS obstacle avoidance node (*rp_obstacle_avoidance*) as described in section 4.2.3.1. In particular, this node's header, implementation and launch files are provided in listings A.1, A.2 and A.3 below.

Listing A.1: Obstacle avoidance node's (*rp_obstacle_avoidance*) header (`/include/obstacle_avoidance.hpp`).

```
/**
 * @author  Manfredas Zabarauskas <manfredas@zabarauskas.com>
 * @date    27/08/2013 19:49:22
 *
 * @class RPObstacleAvoidanceNode
 *
 * @brief Robot photographer's obstacle avoidance ROS node, which uses point cloud and depth image
 *        inputs to detect obstacles in front of the robot, and generates the driving directions
 *        accordingly.
 */
#ifndef OBSTACLE_AVOIDANCE_HPP_
#define OBSTACLE_AVOIDANCE_HPP_

// STL includes
#include <deque>

// ROS includes
#include <ros/ros.h>
#include <sensor_msgs/PointCloud2.h>
#include <std_msgs/Float64.h>
#include <std_msgs/UInt8.h>

// PCL includes
#include <pcl/ros/conversions.h>
#include <pcl_ros/point_cloud.h>
#include <pcl/point_types.h>

// Boost includes
#include <boost/thread/mutex.hpp>
```

```
// RPLocomotion includes
#include <driving_direction.h>

// Constants
#define DRIVING_DIRECTION_TOPIC  "/rp/obstacle_avoidance/driving_direction"
#define SENSOR_TILT_ANGLE_TOPIC  "/cur_tilt_angle"
#define SENSOR_POINT_CLOUD_TOPIC "/camera/depth_registered/points"
#define SENSOR_DEPTH_IMAGE_TOPIC "/camera/depth_registered/image"

#define SENSOR_DISTANCE_FROM_GROUND   0.61f

// Overridable parameter defaults
#define FOCUS_FIELD_WIDTH_DEFAULT      0.4
#define FOCUS_FIELD_HEIGHT_DEFAULT     1.4
#define FOCUS_FIELD_DEPTH_DEFAULT      0.5
#define CLOUD_FILTER_SIZE_DEFAULT      0.03
#define MAX_INVALID_DEPTH_DATA_DEFAULT 0.4
#define SMOOTHING_FRAME_LIMIT_DEFAULT  2
#define VERBOSE_OUTPUT_ENABLED_DEFAULT false


class RPObstacleAvoidanceNode
{
   private:
      ros::NodeHandle& node;                      /**< Node handle.                          */

      DrivingDirection current_direction;         /**< Robot's current driving direction.    */

      std::deque<unsigned> point_counts_ahead;    /**< History of points ahead of robot.     */

      float sensor_angle;                         /**< Sensor's angle.                       */
      boost::mutex sensor_angle_mutex;            /**< Sensor's angle lock.                  */

      ros::Publisher driving_direction_publisher; /**< Driving direction publisher.          */

      double FOCUS_FIELD_WIDTH;                   /**< ROI width overridable parameter (OP). */
      double FOCUS_FIELD_HEIGHT;                  /**< ROI height OP.                        */
      double FOCUS_FIELD_DEPTH;                   /**< ROI depth OP.                         */
      double CLOUD_FILTER_SIZE;                   /**< Voxel filter grid size OP.            */
      double MAX_INVALID_DEPTH_DATA;              /**< Maximum allowed percentage OP.        */
      int    SMOOTHING_FRAME_LIMIT;               /**< Smoothing frame limit OP.             */
      bool   VERBOSE_OUTPUT_ENABLED;              /**< Verbose output OP.                    */

      /**
       * Reduces the number of points in the cloud using voxel grid filter.
       * @param cloud_to_filter Point cloud to be filtered.
       * @param filtered_cloud  Resulting filtered point cloud.
       * @param voxel_size      Voxel grid filter size.
       */
      void reducePointCloudDensity(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud_to_filter
                                   pcl::PointCloud<pcl::PointXYZ>::Ptr& filtered_cloud,
                                   double voxel_size);

      /**
       * Levels the point cloud w.r.t. the ground (using Kinect's accelerometer).
       * @param cloud_to_level Point cloud to be levelled.
       * @param leveled_cloud  Resulting levelled point cloud.
       */
      void levelPointCloud(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud_to_level,
                           pcl::PointCloud<pcl::PointXYZ>::Ptr& levelled_cloud);

      /**
       * Reduces the point cloud to an area of interest (ROI) in which robot looks for obstacles.
       * @param cloud_to_crop  Point cloud to be cropped.
       * @param cropped_cloud  Resulting cropped point cloud.
       * @param x_limit_left   X axis left crop limit.
       * @param x_limit_right  X axis right crop limit.
       * @param y_limit_above  Y axis top crop limit.
       * @param y_limit_below  Y axis bottom crop limit.
       * @param z_limit_ahead  Z axis front crop limit.
       * @param z_limit_behind Z axis back crop limit.
       */
      void cropPointCloud(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud_to_crop,
                          pcl::PointCloud<pcl::PointXYZ>::Ptr& cropped_cloud,
                          double x_limit_left,
                          double x_limit_right,
                          double y_limit_above,
                          double y_limit_below,
                          double z_limit_ahead,
                          double z_limit_behind);

      /**
       * Gets the proportion of invalid data (missing depth readings) in a given depth image.
       * @param depth_image Input depth image.
       * @returns Proportion of invalid data in the given depth image (between 0.0 and 1.0).
       */
      double getInvalidDepthDataProportion(const sensor_msgs::Image::ConstPtr& depth_image);

      /**
       * Computes the driving direction from the levelled point cloud in front of the robot.
       * @param frontal_point_cloud Levelled point cloud in front of the robot.
       * @returns Computed driving direction.
       */
      DrivingDirection drivingDirectionFromFrontalPointCloud(
          const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& frontal_point_cloud);

      /**
       * Publishes the given driving direction.
       * @param driving_direction Driving direction to publish.
       */
      void publishDrivingDirection(const DrivingDirection driving_direction);

      /**
       * Gets the overridable parameters from the parameter server.
       */
      void getOverridableParameters();

      /**
       * Callback for the point cloud and depth image inputs from the Kinect.
       * @param point_cloud Input point cloud from Kinect.
       * @param depth_image Input depth image from Kinect.
       */
      void kinectInputCallback(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& point_cloud,
                               const sensor_msgs::Image::ConstPtr& depth_image);

      /**
       * Callback for the accelerometer's inputs from the Kinect.
       * @param angle Kinect's tilt angle.
       */
      void sensorAngleCallback(const std_msgs::Float64& angle);

   public:
      /**
       * Default obstacle avoidance node's constructor.
       * @param node Handle to ROS node.
       */
      RPObstacleAvoidanceNode(ros::NodeHandle& node);
};

#endif /* OBSTACLE_AVOIDANCE_HPP_ */
```

Listing A.2: Obstacle avoidance node's (*rp_obstacle_avoidance*) implementation (/src/obstacle_avoidance.cpp).

```
/**
 * @author  Manfredas Zabarauskas <manfredas@zabarauskas.com>
 * @date    27/08/2013 19:49:22
 */
#include <obstacle_avoidance.hpp>

// STL includes
#include <iostream>
#include <cmath>

// ROS includes
#include <message_filters/subscriber.h>
#include <message_filters/synchronizer.h>
#include <message_filters/sync_policies/approximate_time.h>
#include <sensor_msgs/Image.h>

// PCL includes
#include <pcl/ros/conversions.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/passthrough.h>
#include <pcl/common/transforms.h>

// Macro to check whether a given raw depth value is invalid (Kinect/GFreenect specific)
#define INVALID_DEPTH_VALUE(ITERATOR) ((*((ITERATOR) + 0) == 0) && \
                                       (*((ITERATOR) + 1) == 0) && \
                                       (*((ITERATOR) + 2) == 192) && \
                                       (*((ITERATOR) + 3) == 127))

int main(int argc, char** argv)
{
    // Initialize ROS
    ros::init(argc, argv, "rp_obstacle_avoidance");

    // Get the handle to the ROS node
    ros::NodeHandle node;

    // Start the worker node
    RPObstacleAvoidanceNode worker_node(node);
}


RPObstacleAvoidanceNode::RPObstacleAvoidanceNode(ros::NodeHandle& node) :
    node(node),
    sensor_angle(0.0f),
    current_direction(STOP),

    // Initialize overridable parameters
    FOCUS_FIELD_WIDTH(FOCUS_FIELD_WIDTH_DEFAULT),
    FOCUS_FIELD_HEIGHT(FOCUS_FIELD_HEIGHT_DEFAULT),
    FOCUS_FIELD_DEPTH(FOCUS_FIELD_DEPTH_DEFAULT),
    CLOUD_FILTER_SIZE(CLOUD_FILTER_SIZE_DEFAULT),
    MAX_INVALID_DEPTH_DATA(MAX_INVALID_DEPTH_DATA_DEFAULT),
    SMOOTHING_FRAME_LIMIT(SMOOTHING_FRAME_LIMIT_DEFAULT),
    VERBOSE_OUTPUT_ENABLED(VERBOSE_OUTPUT_ENABLED_DEFAULT)
{
    // Get parameters from the parameter server
    getOverridableParameters();

    // Initialize ROS publisher for the driving direction
    const int DRIVING_DIRECTION_BUFFER_SIZE = 1;
    driving_direction_publisher = node.advertise<std_msgs::UInt8>(
        DRIVING_DIRECTION_TOPIC,
        DRIVING_DIRECTION_BUFFER_SIZE
    );
    // Initialize ROS subscriber for the Kinect sensor angle
    const int SENSOR_TILT_ANGLE_BUFFER_SIZE = 1;
    node.subscribe(SENSOR_TILT_ANGLE_TOPIC,
                   SENSOR_TILT_ANGLE_BUFFER_SIZE,
                   &RPObstacleAvoidanceNode::sensorAngleCallback,
                   this);

    // Initialize synchronized ROS subscribers for the Kinect's point cloud input and depth images
    const int SYNCHRONIZED_SUBSCRIBERS_BUFFER_SIZE = 5;
    message_filters::Subscriber<pcl::PointCloud<pcl::PointXYZ> > point_cloud_subscriber(
        node,
        SENSOR_POINT_CLOUD_TOPIC,
        SYNCHRONIZED_SUBSCRIBERS_BUFFER_SIZE
    );

    message_filters::Subscriber<sensor_msgs::Image> depth_data_subscriber(
        node,
        SENSOR_DEPTH_IMAGE_TOPIC,
        SYNCHRONIZED_SUBSCRIBERS_BUFFER_SIZE
    );

    // Create a synchronized subscriber for Kinect's point cloud and depth image inputs
    typedef message_filters::sync_policies::ApproximateTime<
       pcl::PointCloud<pcl::PointXYZ>, sensor_msgs::Image
    > SynchronizationPolicy;

    const int SYNCHRONIZATION_WINDOW_SIZE = 10;
    message_filters::Synchronizer<SynchronizationPolicy> synchronized_subscriber(
        SynchronizationPolicy(SYNCHRONIZATION_WINDOW_SIZE),
        point_cloud_subscriber,
        depth_data_subscriber
    );

    // Register the callback for the synchronized subscriber
    synchronized_subscriber.registerCallback(
        boost::bind(&RPObstacleAvoidanceNode::kinectInputCallback, this, _1, _2)
    );

    // Spin the node at 5 Hz
    ros::Rate rate(5);
    while (ros::ok())
    {
        ros::spinOnce();
        rate.sleep();
    }
};


void RPObstacleAvoidanceNode::kinectInputCallback(
    const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& point_cloud,
    const sensor_msgs::Image::ConstPtr& depth_image)
{
    // Check the proportion of valid depth readings; if the depth image is nearly empty, we must
    // be standing in front of a large object.
    DrivingDirection proposed_direction;
    if (getInvalidDepthDataProportion(depth_image) > MAX_INVALID_DEPTH_DATA)
    {
        proposed_direction = (current_direction != FORWARD) ? current_direction : RIGHT;
    }
    else
    {
        // Create an empty point cloud
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered(new pcl::PointCloud<pcl::PointXYZ>);

        // Reduce the point cloud density
        reducePointCloudDensity(point_cloud, cloud_filtered, CLOUD_FILTER_SIZE);
        // Level it to be parallel to the ground
        levelPointCloud(cloud_filtered, cloud_filtered);

        // Crop it to the area in front of the robot
        cropPointCloud(cloud_filtered,
                       cloud_filtered,
                      -FOCUS_FIELD_WIDTH / 2,
                       FOCUS_FIELD_WIDTH / 2,
                      -FOCUS_FIELD_HEIGHT,
                       0.0,
                       FOCUS_FIELD_DEPTH,
                       0.0);

        // Get the driving direction from the contents of the point cloud ahead of the robot
        proposed_direction = drivingDirectionFromFrontalPointCloud(cloud_filtered);
    }

    // Take care to ensure that the current turn direction is maintained (to avoid oscillation)
    if (current_direction == FORWARD || proposed_direction == FORWARD)
    {
        if (proposed_direction != current_direction)
        {
            publishDrivingDirection(proposed_direction);
            current_direction = proposed_direction;
        }
    }

    // Produce output if necessary
    if (VERBOSE_OUTPUT_ENABLED)
    {
        ROS_INFO("Current direction: %s", ((current_direction == FORWARD) ? "FORWARD" :
                                           (current_direction == LEFT) ? "LEFT" : "RIGHT"));
    }
};


void RPObstacleAvoidanceNode::levelPointCloud(
    const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud_to_level,
    pcl::PointCloud<pcl::PointXYZ>::Ptr& levelled_cloud)
{
    // Get the last known sensor's tilt angle
    float angle_rad;
    sensor_angle_mutex.lock();
    {
        angle_rad = M_PI * sensor_angle / 180.0f;
    }
    sensor_angle_mutex.unlock();

    // Create the appropriate rotation matrix
    Eigen::Matrix4f rotation_matrix;
    rotation_matrix <<
        1.0f,     0.0f,            0.0f,            0.0f,
        0.0f,     cos(angle_rad), -sin(angle_rad), -SENSOR_DISTANCE_FROM_GROUND,
        0.0f,     sin(angle_rad), cos(angle_rad),  0.0f,
        0.0f,     0.0f,            0.0f,            1.0f;

    // Rotate back the point cloud according to input from Kinect's accelerometer
    pcl::transformPointCloud(*cloud_to_level, *levelled_cloud, rotation_matrix);
}


DrivingDirection RPObstacleAvoidanceNode::drivingDirectionFromFrontalPointCloud(
    const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& frontal_point_cloud)
{
    // Do not use smoothing while turning
    if (current_direction != FORWARD)
    {
        point_counts_ahead.clear();
    }

    // Add the sample of the number of points ahead, capping the sample count
    point_counts_ahead.push_front(frontal_point_cloud->size());
    while (point_counts_ahead.size() > SMOOTHING_FRAME_LIMIT)
    {
        point_counts_ahead.pop_back();
    }

    // Get the smoothed number of points ahead
    int smoothed_point_count_ahead = 0;
    for (unsigned i = 0; i < point_counts_ahead.size(); i++)
    {
        smoothed_point_count_ahead += point_counts_ahead[i];
    }
    smoothed_point_count_ahead /= point_counts_ahead.size();

    if (smoothed_point_count_ahead == 0)
    {
        // Coast is clear
        return FORWARD;
    }
    else
    {
        // Find out on which side of the robot (left/right) lies the centroid of an obstacle and
        // turn in the opposite direction,
        float centroid_x = 0.0f;
        for (unsigned i = 0; i < frontal_point_cloud->size(); i++)
        {
            centroid_x += frontal_point_cloud->points[i].x;
        }
        centroid_x /= frontal_point_cloud->size();

        return (centroid_x < 0.0) ? RIGHT : LEFT;
    }
}


void RPObstacleAvoidanceNode::reducePointCloudDensity(
    const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud_to_filter,
    pcl::PointCloud<pcl::PointXYZ>::Ptr& filtered_cloud,
    double voxel_size)
{
    // Create the appropriate voxel grid filter
    pcl::VoxelGrid<pcl::PointXYZ> voxel_size_filter;
    voxel_size_filter.setInputCloud(cloud_to_filter);
    voxel_size_filter.setLeafSize(voxel_size, voxel_size, voxel_size);

    // Subsample the input point cloud
    voxel_size_filter.filter(*filtered_cloud);
};


void RPObstacleAvoidanceNode::cropPointCloud(
    const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& cloud_to_crop,
    pcl::PointCloud<pcl::PointXYZ>::Ptr& cropped_cloud,
    double x_limit_left,
    double x_limit_right,
    double y_limit_above,
    double y_limit_below,
    double z_limit_ahead,
    double z_limit_behind)
{
    // Create the appropriate pass-through filter
    pcl::PassThrough<pcl::PointXYZ> pass_through_filter;
    pass_through_filter.setInputCloud(cloud_to_crop);

    // Crop horizontally
    pass_through_filter.setFilterFieldName("x");
    pass_through_filter.setFilterLimits(x_limit_left, x_limit_right);
    pass_through_filter.filter(*cropped_cloud);

    // Crop vertically
    pass_through_filter.setFilterFieldName("y");
    pass_through_filter.setFilterLimits(y_limit_above, y_limit_below);
    pass_through_filter.filter(*cropped_cloud);

    // Crop depth-wise
    pass_through_filter.setFilterFieldName("z");
    pass_through_filter.setFilterLimits(z_limit_behind, z_limit_ahead);
    pass_through_filter.filter(*cropped_cloud);
};


double RPObstacleAvoidanceNode::getInvalidDepthDataProportion(
    const sensor_msgs::Image::ConstPtr& depth_image)
{
    int invalid_depth_pixels = 0;

    // Count the invalid depth pixels directly in raw depth data (in 32FC1 encoding)
    for (std::vector<unsigned char>::const_iterator iterator = depth_image->data.begin(),
         iterator_end = depth_image->data.end();
         iterator != iterator_end;
         std::advance(iterator, 4))
    {
        invalid_depth_pixels += INVALID_DEPTH_VALUE(iterator);
    }

    return 4.0 * (double)invalid_depth_pixels / (double)depth_image->data.size();
}


void RPObstacleAvoidanceNode::publishDrivingDirection(const DrivingDirection driving_direction)
{
    // Convert and publish the direction
    std_msgs::UInt8 converted_direction;
    converted_direction.data = drivingDirectionToUint8(driving_direction);

    driving_direction_publisher.publish(converted_direction);
}


void RPObstacleAvoidanceNode::sensorAngleCallback(const std_msgs::Float64& angle)
{
    // Save sensor's angle
    sensor_angle_mutex.lock();
    {
        sensor_angle = angle.data;
    }
    sensor_angle_mutex.unlock();
}


void RPObstacleAvoidanceNode::getOverridableParameters()
{
    // Get overridable parameters from the parameter server
    node.getParamCached("/rp/obstacle_avoidance_node/focus_field_width",     FOCUS_FIELD_WIDTH);
    node.getParamCached("/rp/obstacle_avoidance_node/focus_field_height",    FOCUS_FIELD_HEIGHT);
    node.getParamCached("/rp/obstacle_avoidance_node/focus_field_depth",     FOCUS_FIELD_DEPTH);
    node.getParamCached("/rp/obstacle_avoidance_node/cloud_filter_size",     CLOUD_FILTER_SIZE);
    node.getParamCached("/rp/obstacle_avoidance_node/smoothing_frame_limit", SMOOTHING_FRAME_LMT);
    node.getParamCached("/rp/obstacle_avoidance_node/max_invalid_depth_data",MAX_NVLD_DEPTH_DATA);
    node.getParamCached("/rp/obstacle_avoidance_node/verbose_output_enabled",VERBOSE_OUT_ENABLED);
}
```

Listing A.3: Obstacle avoidance node's (*rp_obstacle_avoidance*) launch file (`/launch/obstacle_avoidance.launch`).

```
<launch>
  <node name="kinect_aux_node" pkg="kinect_aux" type="kinect_aux_node"/>
  <node ns="rp" name="obstacle_avoidance_node" pkg="rp_obstacle_avoidance"
        type="rp_obstacle_avoidance_node">

    <!-- Node enabled flag -->
    <param name="enabled" value="true" type="bool"/>

    <!-- Verbose output enabled flag -->
    <param name="verbose_output_enabled" value="false" type="bool"/>

    <!-- ROI width (meters) -->
    <param name="focus_field_width" value="0.45" type="double"/>

    <!-- ROI height (meters) -->
    <param name="focus_field_height" value="1.4" type="double"/>

    <!-- ROI depth (meters) -->
    <param name="focus_field_depth" value="0.7" type="double"/>

    <!-- Voxel filter grid size (meters) -->
    <param name="cloud_filter_size" value="0.05" type="double"/>

    <!-- Smoothing frame limit (increases resistance to noise, but decreases responsiveness) -->
    <param name="smoothing_frame_limit" value="3" type="int"/>

    <!-- Maximum allowed percentage of invalid depth data in the depth image -->
    <param name="max_invalid_depth_data" value="0.4" type="double"/>
  </node>
</launch>
```

Bibliography

R. Achanta, S. Hemami, F. Estrada, and S. Sabine. Frequency-Tuned Salient Region Detection. In *Proceedings of the 2009 IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1597–1604, 2009.

J. Agbinya and D. Rees. Multi-Object Tracking in Video. *Real-Time Imaging*, 5(5):295–304, 1999.

H. Ahn, D. Kim, J. Lee, S. Chi, K. Kim, J. Kim, M. Hahn, and H. Kim. A Robot Photographer with User Interactivity. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5637–5643, 2006.

S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

C. Barbé, H. Figuière, H. U. Niedermann, M. Meissner, and S. Fritzinger. gPhoto2: Digital Camera Software, 2002. URL: `http://www.gphoto.org`. Last accessed on 20/08/2013.

F. Basso. *RGB-D People Tracking by Detection for a Mobile Robot*. Master's thesis, University of Padova, 2011.

F. Basso, M. Munaro, S. Michieletto, E. Pagello, and E. Menegatti. Fast and Robust Multi-people Tracking from RGB-D Data for a Mobile Robot. In *Proceedings of the 12th Intelligent Autonomous Systems (IAS) Conference*, pages 265–276, 2012.

F. Bonin-Font, A. Ortiz, and G. Oliver. Visual Navigation for Mobile Robots: A Survey. *Journal of Intelligent & Robotic Systems*, 53(3):263–296, 2008.

J. Borenstein and Y. Koren. Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.

G. Borgefors. Distance Transformations in Digital Images. *Computer Vision, Graphics, and Image Processing*, 34(3): 344–371, June 1986.

L. Bottou and C.-J. Lin. Support Vector Machine Solvers. In *Large Scale Kernel Machines*, pages 301–320. MIT Press, 2007.

S. Boucher. Obstacle Detection and Avoidance using TurtleBot Platform and XBox Kinect. Technical report, Rochester Institute of Technology, 2012.

J.-Y. Bouguet. Camera Calibration Toolbox for Matlab, 2010. URL: `http://www.vision.caltech.edu/bouguetj/calib_doc/`. Last accessed on 19/08/2013.

G. R. Bradski. Computer Vision Face Tracking For Use in a Perceptual User Interface. *Intel Technology Journal*, (Q2):214–219, 1998.

G. R. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 25(11):120, 122–125, 2000.

C. Braillon, K. Usher, C. Pradalier, J. L. Crowley, and C. Laugier. Fusion of Stereo and Optical Flow Data using Occupancy Grids. In *Proceedings of the 2006 Intelligent Transportation Systems Conference*, pages 1240–1245, 2006.

L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

R. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1): 14–23, 1986.

D. C. Brown. Decentering Distortion of Lenses. *Photometric Engineering*, 32(3):444–462, 1966.

D. C. Brown. Close-Range Camera Calibration. *Photogrammetric Engineering*, 37(8):855–866, 1971.

M. B. Brown and A. B. Forsythe. The Small Sample Behavior of Some Statistics Which Test the Equality of Several Means. *Technometrics*, 16(1):129–132, 1974a.

M. B. Brown and A. B. Forsythe. Robust Tests for Equality of Variances. *Journal of the American Statistical Association*, 69:364–367, 1974b.

D. R. Bueno, E. Viruete, and L. Montano. An Autonomous Tour Guide Robot in a Next Generation Smart Museum. In *Proceedings of the 5th International Symposium on Ubiquitous Computing and Ambient Intelligence*, 2011.

Z. Byers, M. Dixon, K. Goodier, C. M. Grimm, and W. D. Smart. An Autonomous Robot Photographer. *Intelligent Robots and Systems*, 3:2636–2641, 2003.

Z. Byers, M. Dixon, W. D. Smart, and C. M. Grimm. Say Cheese! Experiences with a Robot Photographer. In *Proceedings of the 15th Innovative Applications of Artificial Intelligence Conference*, volume 25, pages 37–46, 2004.

J. Campbell and P. Pillai. Leveraging Limited Autonomous Mobility to Frame Attractive Group Photos. In *IEEE International Conference on Robotics and Automation*, pages 3396–3401, 2005.

J. F. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

J. Carifio and R. Perla. Resolving the 50-year Debate Around Using and Misusing Likert Scales. *Medical Education*, 42:1150–1152, 2008.

G. Chavers, B. Ballard, B. A. Cohen, T. McGee, J. Moore, C. Reed, and C. Stemple. Robotic Lunar Lander Development Status. Technical report, NASA, 2012.

B. Choi, C. Meriçli, J. Biswas, and M. Veloso. Fast Human Detection for Indoor Mobile Robots using Depth Images. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

W. Choi, C. Pantofaru, and S. Savarese. Detecting and Tracking People using an RGB-D Camera via Multiple Detector Fusion. In *Workshop on Challenges and Opportunities in Robot Perception, at the International Conference on Computer Vision (ICCV)*, pages 1076–1083, 2011.

L.-Y. Chung. Remote Teleoperated and Autonomous Mobile Security Robot Development in Ship Environment. *Mathematical Problems in Engineering*, page 14, 2013.

J. Cohen. Weighted Kappa: Nominal Scale Agreement Provision for Scaled Disagreement or Partial Credit. *Psychological Bulletin*, 70(4):213–220, 1968.

D. Comaniciu and P. Meer. Mean Shift: A Robust Approach toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

G. Cybenko. Approximations by Superpositions of Sigmoidal Functions. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pages 886–893, 2005.

N. X. Dao, B.-J. You, and S.-R. Oh. Visual Navigation for Indoor Mobile Robots using a Single Camera. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1992–1997, 2005.

Z. De-An, L. Jidong, J. Wei, Z. Ying, and C. Yu. Design and Control of an Apple Harvesting Robot. *Biosystems Engineering*, 110(2):112–122, Oct. 2011.

G. DeSouza and A. Kak. Vision for Mobile Robot Navigation: a Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002.

R. Dias, A. J. R. Neves, J. L. Azevedo, B. Cunha, J. Cunha, P. Dias, A. Domingos, L. Ferreira, P. Fonseca, N. Lau, E. Pedrosa, A. Pereira, R. Serra, J. Silva, P. Soares, and A. Trifan. CAMBADA' 2013: Team Description Paper. Technical report, RoboCup, 2013.

M. A. Diftler, J. S. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. A. Permenter, B. K. Hargrave, R. Platt, R. T. Savely, R. O. Ambrose, G. Motors, and O. S. Systems. Robonaut 2 - The First Humanoid Robot in Space. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 2178–2183, 2011.

M. Dixon, C. Grimm, and W. Smart. Picture Composition for a Robot Photographer. Technical report, Washington University in St. Louis, 2003.

P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–61, Apr. 2012.

I. Dryanovski, W. Morris, S. Magnenat, R. B. Rusu, and P. Mihelich. Kinect AUX Driver for ROS, 2011. URL: `http://www.ros.org/wiki/kinect_aux`. Last accessed on 30/08/2013.

R. O. Duda and P. E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM*, 15(1):11–15, 1972.

J. Duddington. eSpeak: Speech Synthesizer, 2006. URL: `http://espeak.sourceforge.net/`. Last accessed on 21/08/2013.

O. J. Dunn. Multiple Comparisons Among Means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

M. Enzweiler and D. M. Gavrila. Monocular Pedestrian Detection: Survey and Experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2179–2195, Dec. 2009.

F. Faber, M. Bennewitz, C. Eppner, A. Görög, C. Gonsior, D. Joho, M. Schreiber, and S. Behnke. The Humanoid Museum Tour Guide Robotinho. In *Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication*, number September, pages 891–896, 2009.

G. Fant. *Acoustic Theory of Speech Production*. Mouton & Co, The Hague, Netherlands, 1960.

D. Feil-Seifer and M. J. Matarić. B3IA: A Control Architecture for Autonomous Robot-Assisted Behavior Intervention for Children with Autism Spectrum Disorders. In *Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication*, pages 328–333, 2008.

R. Feraund, O. Bernier, J.-E. Viallet, and M. Collobert. A Fast and Accurate Face Detector Based on Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(1):42–53, 2001.

M. A. Fischler and R. C. Bolles. Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

R. A. Fisher. On a Distribution Yielding the Error Functions of Several Well-Known Statistics. In *Proceedings of the International Congress of Mathematicians*, pages 805–813, 1924.

C. Fitzgerald. Developing Baxter. In *Proceedings of the 2013 IEEE International Conference on Technologies for Practical Robot Applications*, pages 1–6, 2013.

M. M. Fleck, D. A. Forsyth, and C. Bregler. Finding Naked People. In *Proceedings of the 1996 European Conference on Computer Vision*, pages 593–602, 1996.

J. Forlizzi and C. Disalvo. Service Robots in the Domestic Environment: A Study of the Roomba Vacuum in the Home. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, pages 258–265, 2006.

D. A. Forsyth and M. M. Fleck. Automatic Detection of Human Nudes. *International Journal of Computer Vision*, 32(1):63–77, 1999.

Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

K. Fukunaga and L. Hostetler. The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

E. F. Fukushima, M. Freese, T. Matsuzawa, T. Aibara, and S. Hirose. Humanitarian Demining Robot Gryphon - Current Status and an Objective Evaluation. *International Journal on Smart Sensing and Intelligent Systems*, 1(3): 735–753, 2008.

K. Fukushima. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36(4):93–202, 1980.

R. Gadde and K. Karlapalem. Aesthetic Guideline Driven Photography by Robots. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 3, pages 2060–2065, 2011.

G. Galatas, C. McMurrough, G. L. Mariottini, and F. Makedon. eyeDog: an Assistive-Guide Robot for the Visually Impaired. In *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, 2011.

C. Gardner. Meet Roborazzi, 2012. URL: `http://www.stuff.co.nz/waikato-times/business/technology/7625893/Meet-Roborazzi`. Last accessed on 29/07/2013.

J. Garstka and G. Peters. View-dependent 3D Projection using Depth-Image-based Head Tracking. In *Proceedings of the 8th IEEE International Workshop on Projector-Camera Systems*, pages 52–58, 2011.

D. M. Gavrila. A Bayesian, Exemplar-Based Approach to Hierarchical Shape Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1408–21, Aug. 2007.

D. M. Gavrila and V. Philomin. Real-Time Object Detection for "Smart" Vehicles. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 87–93, 1999.

P. F. Gomes, E. M. Segura, H. Cramer, A. Paiva, T. Paiva, and L. E. Holmquist. ViPleo and PhyPleo: Artificial Pet with Two Embodiments. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, 2011.

Google. Google C++ Style Guide, 2013. URL: `http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml`. Last accessed on 30/08/2013.

W. S. Gosset. The Probable Error of the Mean. *Biometrika*, 6(1):1–25, 1908.

D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. The NAO Humanoid: a Combination of Performance and Affordability. In *Computing Research Repository*, pages 1–10, 2008.

T. Graber, S. Kohlbrecher, J. Meyer, K. Petersen, O. von Stryk, and U. Klingauf. Robot League Team Hector Darmstadt. Technical report, RoboCupRescue, 2013.

B. Graf, U. Reiser, M. Hägele, K. Mauz, and P. Klein. Robotic Home Assistant Care-O-bot 3 - Product Vision and Innovation Platform. *IEEE Workshop on Advanced Robotics and its Social Impacts*, pages 139–144, 2009.

T. Grill and M. Scanlon. *Photographic Composition*. American Photographic Book Publishing, Orlando, FL, 1990.

J. P. Grotzinger, J. Crisp, A. R. Vasavada, R. C. Anderson, C. J. Baker, R. Barry, D. F. Blake, P. Conrad, K. S. Edgett, B. Ferdowski, R. Gellert, J. B. Gilbert, M. Golombek, J. Gómez-Elvira, D. M. Hassler, L. Jandura, M. Litvak, P. Mahaffy, J. Maki, M. Meyer, M. C. Malin, I. Mitrofanov, J. J. Simmonds, D. Vaniman, R. V. Welch, and R. C. Wiens. Mars Science Laboratory Mission and Science Investigation. *Space Science Reviews*, 170(1-4):5–56, July 2012.

A. Hadid, M. Pietikäinen, and T. Ahonen. A Discriminative Feature Space for Detecting and Recognizing Faces. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2(II): 797–804, 2004.

C. Harris and M. Stephens. A Combined Corner and Edge Detector. In M. M. Matthews, editor, *Proceedings of the 4th Alvey Vision Conference*, volume 15, pages 147–151. Manchester, UK, Manchester, UK, 1988.

F. Hegger, N. Hochgeschwender, G. K. Kraetzschmar, and P. G. Ploeger. People Detection in 3D Point Clouds using Local Surface Normals. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 154–165. Springer Berlin Heidelberg, 2012.

B. Heisele, T. Serre, M. Pontil, and T. Poggio. Component-Based Face Detection. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, pages 657–662, 2001.

D. Holz, S. Holzer, R. B. Rusu, and S. Behnke. Real-Time Plane Segmentation using RGB-D Cameras. In *RoboCup Symposium*, 2011.

R. Hoogendijk, G. J. L. Naus, F. B. F. Schoenmakers, and C. A. Lopez. Tech United Eindhoven Team Description 2012. Technical report, RoboCup, 2012.

G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. *University of Massachusetts Amherst Technical Report 07*, 49 (07-49):1–11, 2007.

M. Hvilshoj, S. Bogh, O. Madsen, and M. Kristiansen. The Mobile Robot "Little Helper": Concepts, Ideas and Working Principles. In *Proceedings of the 2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–4, 2009.

Innvo Labs. PLEOworld, 2013. URL: `http://www.pleoworld.com`. Last accessed on 30/07/2013.

S. Jamieson. Likert Scales: How to (Ab)use Them. *Medical Education*, 38(12):1217–1218, 2004.

B. Janko, C. P. Turner, K. S. Cave-Ayland, M. K. Pittuck, S. O. Madgwick, and W. S. Harwin. A Multiple Robot Solution for Urban Reconnaissance. *International Journal of Intelligent Defence Support Systems*, 4(1):70, 2011.

S. Jin, D. Kim, H. S. Kim, C. H. Lee, J. S. Choi, and J. W. Jeon. Real-Time Sound Source Localization System Based on FPGA. In *Proceedings of the 6th IEEE International Conference on Industrial Informatics*, number 1, pages 673–677, July 2008.

M. Jones and J. M. Rehg. Statistical Color Models with Application to Skin Detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.

P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A Survey of Skin-Color Modeling and Detection Methods. *Pattern Recognition*, 40(3):1106–1122, Mar. 2007.

Z. Khan, T. Balch, and F. Dellaert. An MCMC-based Particle Filter for Tracking Multiple Interacting Targets. In *Proceedings of the 2004 European Conference on Computer Vision*, pages 279–290. Springer Berlin Heidelberg, 2004.

M.-J. Kim, T.-H. Song, S.-H. Jin, S.-M. Jung, G.-H. Go, K.-H. Kwon, and J.-W. Jeon. Automatically Available Photographer Robot for Controlling Composition and Taking Pictures. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6010–6015, 2010.

R. E. Kirk. *Experimental Design: Procedures For The Behavioral Sciences*. Brooks/Cole, Pacific Grove, CA, USA, 3rd edition, 1995.

S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598): 671–680, 1983.

D. H. Klatt. Software for a Cascade/Parallel Formant Synthesizer. *Journal of the Acoustical Society of America*, 67 (3):971–995, 1980.

W. H. Kruskal and A. W. Wallis. Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

E. H. Land and J. J. McCann. Lightness and Retinex Theory. *Journal of the Optical Society of America*, 61(1):1–11, 1971.

B. Leibe, E. Seemann, and B. Schiele. Pedestrian Detection in Crowded Scenes. In *Proceedings of the 2005 International Conference on Computer Vision and Pattern Recognition*, pages 878–885, 2005.

A. Leykin. Vanishing Points, 2006. URL: `http://www.cs.indiana.edu/~sjohnson/irrt/src/index.htm`. Last accessed on 02/09/2013.

R. Lienhart and J. Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. In *Proceedings of the 2002 International Conference on Image Processing*, volume 1, pages 900–903, 2002.

R. Likert. A Technique for the Measurement of Attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

L. M. Lorigo, R. A. Brooks, and W. E. L. Grimson. Visually-Guided Obstacle Avoidance in Unstructured Environments. In *Proceedings of the 1997 IEEE/RSJ Conference on Intelligent Robots and Systems*, pages 373–379, 1997.

D. G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the 7th International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, volume 2, pages 674–679. Citeseer, Citeseer, 1981.

Y. Luo and X. Tang. Photo and Video Quality Evaluation: Focusing on the Subject. In *Proceedings of the 10th European Conference on Computer Vision: Part III*, pages 386–399, 2008.

J. Maja, T. Takahashi, Z. Wang, and E. Nakano. Real-Time Obstacle Avoidance Algorithm for Visual Navigation. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 925–930, 2000.

S. Maji, A. Berg, and J. Malik. Classification using Intersection Kernel Support Vector Machines is Efficient. *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

H. B. Mann and D. R. Whitney. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947.

J. Mercer. Functions of Positive and Negative Type and Their Connection with the Theory of Integral Equations. *Philosophical Transactions of the Royal Society*, 209(559):415–446, 1909.

K. Mikolajczyk, C. Schmid, and A. Zisserman. Human Detection Based on a Probabilistic Assembly of Robust Part Detectors. In *Proceedings of the 2004 European Conference on Computer Vision*, pages 69–82. Springer Berlin Heidelberg, 2004.

R. Mojtahedzadeh. *Robot Obstacle Avoidance using the Kinect*. Master's thesis, Royal Institute of Technology, Stockholm, Sweden, 2011. URL: `http://kiosk.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2011/rapporter11/mojtahedzadeh_rasoul_11107.pdf`.

D. B. Monro. *Homer: Iliad, Books XIII-XXIV, with Notes*. Oxford Clarendon Press, 4th edition, 1903.

H. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. Doctoral thesis, Carnegie Mellon University, 1980.

S. Munder and D. M. Gavrila. An Experimental Study on Pedestrian Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1863–8, Nov. 2006.

S. Munder, C. Schnörr, and D. M. Gavrila. Pedestrian Detection and Tracking Using a Mixture of View-Based Shape-Texture Models. *IEEE Transactions on Intelligent Transportation Systems*, 9(2):333–343, 2008.

J. Needham. *Science and Civilisation in China: Volume II, History of Scientific Thought*. Cambridge University Press, 1991.

A. Y. Ng and M. I. Jordan. On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes. In *Proceedings of the 2001 Neural Information Processing Systems Conference*, pages 841–848, 2001.

V.-D. Nguyen. Obstacle Avoidance using the Kinect. Technical report, Ho Chi Minh City University of Technology, Vietnam, 2012.

A. Niculescu, B. V. Dijk, A. Nijholt, and D. K. Limbu. Socializing with Olivia, the Youngest Robot Receptionist Outside the Lab. In *Proceedings of the 2nd International Conference on Social Robotics*, pages 50–62, 2010.

N. J. Nilsson. Shakey the Robot. *SRI AI Center Technical Note*, (April), 1984.

G. Norman. Likert Scales, Levels of Measurement and the "Laws" of Statistics. *Advances in Health Sciences Education: Theory and Practice*, 15(5):625–632, 2010.

T. Ojala, M. Pietikäinen, and D. Harwood. A Comparative Study of Texture Measures with Classification Based on Featured Distributions. *Pattern Recognition*, 29(1):51–59, 1996.

T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

OpenKinect. GFreenect Library, 2012. URL: `http://openkinect.org/wiki/GLib_Wrapper`. Last accessed on 30/08/2013.

J. Osada, S. Ohnaka, and M. Sato. The Scenario and Design Process of Childcare Robot, PaPeRo. In *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, 2006.

E. Osuna, R. Freund, and F. Girosit. Training Support Vector Machines: an Application to Face Detection. In *Proceedings of the 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 130–136. IEEE Comput. Soc, 1997.

C. Papageorgiou and T. Poggio. A Trainable System for Object Detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

N. Pears and L. Bojian. Ground Plane Segmentation for Mobile Robot Visual Navigation. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1513–1518, 2001.

K. Pearson. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2(11): 559–572, 1901.

B. Peasley and S. Birchfield. Real-Time Obstacle Detection and Avoidance in the Presence of Specular Surfaces using an Active 3D Sensor. In *2013 IEEE Workshop on Robot Vision (WORV)*, pages 197–202, Jan. 2013.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

M. Peris. Talos and Daedalus: A Review of the Authorship of the Abominable Bronze Man. *The Ceylon Journal of Humanities*, 2(2):29–57, 1971.

T. C. Pham, X. D. Pham, D. D. Nguyen, S. H. Jin, and J. W. Jeon. Dual Hand Extraction using Skin Color and Stereo Information. In *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, pages 330–335, Feb. 2009.

S. L. Phung, A. Bouzerdoum, and D. Chai. Skin Segmentation using Color Pixel Classification: Analysis and Comparison. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(1):148–154, Jan. 2005.

J. Platt. Probabilities for Support Vector Machines. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 2000.

M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an Open-Source Robot Operating System. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, 2009.

Rethink Robotics. Baxter Research Robot - Corporate Brochure, 2013. URL: `http://www.rethinkrobotics.com/index.php/download_file/view/45/262/`. Last accessed on 29/07/2013.

M. Riedmiller. Rprop - Description and Implementation Details. Technical report, University of Karlsruhe, 1994.

H. A. Rowley, S. Baluja, and T. Kanade. Neural Network-Based Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

R. B. Rusu and S. Cousins. 3D is Here: Point Cloud Library (PCL). In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, pages 1–4, Shanghai, China, 2011.

P. Sabzmeydani and G. Mori. Detecting Pedestrians by Learning Shapelet Features. In *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

J. Santos-Victor and G. Sandini. Visual-Based Obstacle Detection: A Purposive Approach using the Normal Flow. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 237–244, 1995.

F. E. Satterthwaite. Synthesis of Variance. *Psychometrika*, 6(5):309–316, 1941.

R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence-Rated Predictions. *Machine Learning. The 11th Annual Conference on Computational Learning Theory*, 37(3):297–336, 1999.

B. Schölkopf, R. Herbrich, and A. J. Smola. A Generalized Representer Theorem. In *Computational Learning Theory*, volume 2111 of *COLT '01/EuroCOLT '01*, pages 416–426. Springer Berlin Heidelberg, 2001.

C. Shen, J. Liu, S. Shih, and J. Hong. Towards Intelligent Photo Composition - Automatic Detection of Unintentional Dissection Lines in Environmental Portrait Photos. *Expert Systems with Applications*, 36(5):9024–9030, July 2009.

H. Shen, T. Lin, S. Chen, and L. Li. Real-Time Seam Tracking Technology of Welding Robot with Visual Sensing. *Journal of Intelligent & Robotic Systems*, 59(3-4):283–298, June 2010.

G. Shirakyan, L.-U. Tan, and H. Davy. Building Roborazzi, a Kinect-Enabled Party Photographer Robot, 2012. URL: `http://channel9.msdn.com/Events/TechEd/NewZealand/TechEd-New-Zealand-2012/INO202`. Last accessed on 02/09/2013.

J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. In *Proceedings of the 24th IEEE Conference on Computer Vision and Pattern Recognition*, pages 1297–1304, June 2011.

R. Simmons, M. Makatchev, R. Kirby, M. K. Lee, I. Fanaswala, B. Browning, J. Forlizzi, and M. Sakr. Believable Robot Characters. *AI Magazine*, 34(4):39–52, 2011.

W. D. Smart, C. M. Grimm, M. Dixon, and Z. Byers. (Not) Interacting with a Robot Photographer. *Proceedings of the AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments. Stanford, CA.*, 2003.

L. Spinello and K. O. Arras. People Detection in RGB-D Data. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3838–3843, Sept. 2011.

J. Stückler and R. Steffens. Real-Time 3D Perception and Efficient Grasp Planning for Everyday Manipulation Tasks. In *Proceedings of the European Conference on Mobile Robots*, pages 177–182, 2011.

S. A. Stüvel. Python Flickr API, 2007. URL: `http://stuvel.eu/flickrapi`. Last accessed on 20/08/2013.

X. Tan and B. Triggs. Enhanced Local Texture Feature Sets for Face Recognition under Difficult Lighting Conditions. *IEEE Transactions on Image Processing*, 19(6):1635–1650, June 2010.

S. Teller, M. R. Walter, M. Antone, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. P. How, A. S. Huang, S. Karaman, B. Luders, N. Roy, and T. Sainath. A Voice-Commandable Robotic Forklift Working Alongside Humans in Minimally-Prepared Outdoor Environments. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, pages 526–533, May 2010.

M. Turk and A. Pentland. Face Recognition Using Eigenfaces. In *Proceedings of the 1991 IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991.

Twitter. Bootstrap Front-End Framework for Faster and Easier Web Development, 2013. URL: `http://getbootstrap.com`. Last accessed on 23/08/2013.

V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, 1998.

P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 511–518, 2001.

P. Viola and M. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.

P. Viola, M. Jones, and D. Snow. Detecting Pedestrians using Patterns of Motion and Appearance. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 734–741, 2003.

X. Wang, T. X. Han, and S. Yan. An HOG-LBP Human Detector with Partial Occlusion Handling. In *Proceedings of the 12th IEEE International Conference on Computer Vision*, pages 32–39, Sept. 2009.

B. L. Welch. The Generalization of "Student's" Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1--2):28–35, 1947.

P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Doctoral thesis, Harvard University, 1974.

C. Wöhler and J. K. Anlauf. An Adaptable Time-Delay Neural-Network Algorithm for Image Sequence Analysis. *IEEE Transactions on Neural Networks*, 10(6):1531–1536, Jan. 1999.

B.-F. Wu, C.-L. Jen, W.-F. Li, T.-Y. Tsou, P.-Y. Tseng, and K.-T. Hsiao. RGB-D Sensor Based SLAM and Human Tracking with Bayesian Framework for Wheelchair Robots. In *Proceedings of the 2013 International Conference on Advanced Robotics and Intelligent Systems*, pages 110–115, May 2013.

S. Wu, S. Yu, and W. Chen. An Attempt to Pedestrian Detection in Depth Images. In *Proceedings of the 3rd Chinese Conference on Intelligent Visual Surveillance*, pages 97–100, 2011.

L. Xia, C.-C. Chen, and J. K. Aggarwal. Human Detection Using Depth Information by Kinect. In *Workshops of the 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 15–22, 2011.

M. H. Yang and N. Ahuja. Detecting Human Faces in Color Images. In *Proceedings of the 1998 IEEE International Conference on Image Processing*, volume 1, pages 127–130, 1998.

M.-H. Yang, D. J. Kriegman, and N. Ahuja. Detecting Faces in Images: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58, 2002.

S. Yu, S. Wu, and L. Wang. SLTP: A Fast Descriptor for People Detection in Depth Images. In *Proceedings of the 9th IEEE International Conference on Advanced Video and Signal-Based Surveillance*, pages 43–47, Sept. 2012.

M. Zabarauskas. *3D Display Simulation Using Head Tracking with Microsoft Kinect*. Bachelor's thesis, University of Cambridge, 2012.

C. Zhang and Z. Zhang. A Survey of Recent Advances in Face Detection. Technical report, Microsoft Research, 2010.

L. Zhang, R. Chu, S. Xiang, S. Liao, and S. Z. Li. Face Detection Based on Multi-Block LBP Representation. In S.-W. Lee and S. Z. Li, editors, *Advances in Biometrics*. Springer Berlin Heidelberg, 2007.

Z. Zhang. Flexible Camera Calibration by Viewing a Plane from Unknown Orientations. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 666–673, 1999.

Z. Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

J. Zhou and B. Li. Homography-based Ground Detection for a Mobile Robot Platform using a Single Camera. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 4100–4105, 2006.