Final Report

B31VL – 6 Month Placement

In partial fulfilment for MEng in Robotics, Autonomous and Interactive Systems

Author: Hazel M Taylor
Matriculation Number: H00150620
Supervisor: Dr Katrin Lohan

Abstract

Robotics for hazardous environments is currently an important area of research, with the ambition of reducing human risk to potentially devastating situations. A cognitive architecture for navigation and goal generation, based on visual perception, for hazardous environments has been proposed and implemented on the robot platform, Cozmo. With limited sensory resources, object detection, self-localisation and mapping, navigation and behaviours were integrated to produce a valid proof of concept for a system capable of assisting humans in potentially hazardous circumstances. This report outlines the work carried out through the duration of my 6-month placement working under the supervision of Dr Katrin Lohan.

Table of Contents

Table of Figures

Table of Tables

Table of Algorithms

ROS - Robot Operating System
HRI - Human Robot Interaction
SLAM - Self-Localisation and Mapping
SIFT - Scale-Invariant Feature Transform
SURF - Speeded Up Robust Features
FAST - Features from Accelerated Segment Test
BRIEF - Binary Robust Independent Elementary Features
ORB - Oriented FAST and Rotated BRIEF
MCL – Monte Carlo Localisation

1 Introduction

1.1 Overview

As the application of robotics is becoming more prevalent in society, many industries deploy robots into hazardous environments to reduce risk to humans. Space, oil and gas, nuclear and construction industries, to name a few, are all faced with potentially dangerous environments where extra support from technological innovations could be valuable. Such systems can increase efficiency and productivity, as well as reduce unnecessary human exposure to conditions such as fumes, hazardous chemicals and potentially fatal circumstances. Primary incident locations such as buildings and enclosed areas are typically the priority in disaster circumstances, with surrounding areas such as streets and towns being of secondary importance. With the support of intelligent systems, tasks such as visual inspections could potentially be carried out around secondary surrounding areas whilst primary locations are dealt with.

Some current applications for robotics in hazardous environments include: decommissioning bombs/explosives, transporting heavy objects and performing mundane tasks more reliably. In all applications, the vital goal is to reduce the demand for humans in dangerous environments. This is currently a key area for research, with many governments, institutions and companies around the world investing in the development of such robotic systems. For example, Heriot-Watt University are currently governing the EPSRC funded ORCA Hub (Offshore Robotics for Certification of Assets) project, with the intention of deploying autonomous intelligent systems onto offshore platforms, aiming to increase safety and decrease the need for personnel [1]. Furthermore, in industry, Taurob [2] develop sophisticated ATEX-certified (certified to work in gas environments without risk of ignition) systems specifically designed for extreme conditions, to carry out tasks such as performing visual inspections, measuring temperatures and navigating through diverse environments e.g. narrow pathways and stairs, to name a few. The energy firm, Total E&P, will make use of their robotic system on their oil drilling rigs from 2020.

Inspection, maintenance and repair are among the key areas in which such mobile robots can contribute to potentially hazardous environments. However, even simple tasks such as patrolling sites for autonomous mobile robots requires multiple components such as mapping, navigation, obstacle avoidance etc. These supply the robot with some form of perception of the environment.

When designing such an intelligent system, the main problems to be considered are as follows:

1. **How can the use of a robot benefit humans in the event of a disaster?**

2. **How can a robot determine whether a route is deemed 'safe' to investigate and navigate accordingly?**

3. **How can a robot detect the nature and location of a hazard and report this?**

If an intelligent system is going to be deployed in the place of a human, it is important that the necessary software is constructed in such a way to effectively sense, perceive and interpret its environment. A cognitive architecture can be used for this. A cognitive architecture can be considered as the underlying infrastructure of an intelligent system [3] and is modelled on the human mind. A general architecture consists of the following key elements [4]:

i. Perception
ii. Attention
iii. Action selection
iv. Memory
v. Learning
vi. Reasoning

These elements are individually considered as research topics in artificial intelligence which can be combined to create an intelligent system. Each of these topics can be implemented in different ways as modules for a bigger system. My interest lies in combining these elements of an architecture and implementing them onto a robot platform to analyse how they perform in practice. Therefore, the overall goal of this project is to develop an adaptive cognitive architecture for autonomous agent behaviour focusing on perception, capable of mapping, self-localising and navigating in real-world inspired hazardous environments with limited resources. Since realistic disaster sites are complex to simulate and the robots currently used in these situations are expensive state-of-the-art research platforms, this project focuses on emulating these tasks on a low-cost consumer platform with limited sensing capabilities, i.e. Cozmo. The developed approach, therefore, also has to be designed with scalability and transferability in mind.

A new outreach project, *RoboCation,* will make use of the proposed architecture as part of their teaching framework. *RoboCation* is a new public engagement venture, to expose robotics to young people, and inspire them to learn about robotics and computer science. *RoboCation* aims to use practical techniques to engage students, giving them hands-on experience working with robots.

This report will begin with a literature review of related work, and the proposed plan for this project. Chapters 4-8 will discuss the design, implementation and results of the work carried out, and the report will conclude with an in-depth discussion and a description of possible future work.

1.2 ROS

The Robot Operating System (ROS) website [5] describes its system as “a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms”. ROS is considered as state-of-the-art in robotics research. It has been adopted by the majority of universities for robotics research, as well as in industry [6]. Hence, ROS supports a wide range of platforms and provides a plethora of software components. This section will briefly describe ROS for the context of this project.

ROS acts as a middleware to connect information gathered from different programs, computers and programming languages. Most components are open source and freely available for research or private use. The main advantage of using ROS is the large community of people contributing and the ability to reuse their code. This means that cutting edge software is made available to the wider community and can therefore be utilised in this project. The main components and tools used for the developed system are listed below.

Roscore is the name server that handles communication between nodes. A *node* is a process that performs computation, i.e. *node* is another word to describe a program. These are designed to be modular, with robotic control systems usually containing multiple nodes. Nodes can communicate with each other via *topics*; they can subscribe to relevant topics to receive data and can publish to topics to share any generated data. *Publishers* and *subscribers* form a ‘data push’ type architecture where call-backs are registered, to deal with new information. There can be multiple subscribers for each topic. For data collection, *rosbag* is a tool that allows to create *bags,* which are a format for saving and playing data from topics; Multiple topics can be recorded at once.

In addition to topics, ROS has *services,* which enables a request-reply interaction, unlike the many-to-many approach of the subscriber and publishers. This remote procedure call cannot be interrupted and should therefore not be used for longer running processes. Another method for moving data usually used to control activities that require longer periods of time is to use *action servers and clients*. These work in a similar way to services, but they can be interrupted, and therefore allow for more flexibility. This is a good structure for handling behaviours, as the client and server communicate via topics and publish feedback whilst running. The figure below shows the general idea for action clients/servers [7].

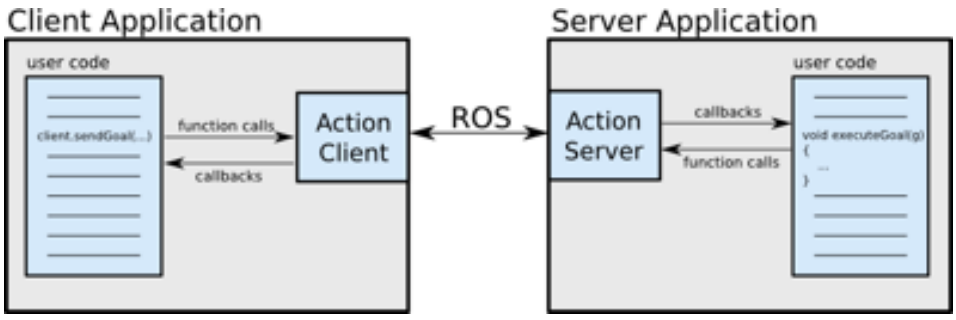


Figure 1 – ROS Action Client/Server diagram

Apart from communication between nodes, ROS also creates a transformation tree which is heavily used in this project. In a transformation tree (*tf tree*), each joint of a robot is represented by a dedicated co-ordinate system. The tf tree, describes how these are connected, i.e. the translation and rotation between them. For example, the base of the robot is called the *base_link* which is connected to the *camera_link.* This connection describes the relative position of the camera to the centre point of the robot, and how it changes over time, e.g. when Cozmo tilts its head. Several of these *frames* build the tf tree.

In summary, as in most robotic systems, the sensor information of Cozmo is streamed on topics which can be subscribed to by higher-level components that interpret the data. One example being the image from the camera that is used for mapping, localisation, and object recognition. The highest level of components for behaviour generation, on the other hand, use the action server/client architecture to start longer running processes (which might use data streamed on topics while running), for example, driving from one waypoint to another, traversing a path consisting of several waypoints, or inspecting valves and gauges.

1.3 Cozmo

Cozmo is a small agent, developed by Anki [8], with few sensors; an IMU in the head (gyroscope and accelerometer), a camera on the 'face', an LED screen for displaying emotions or text, an infrared cliff sensor for edge detection, two tyre tracks for movement and a 'lift' for arms. Compared to state-of-the-art research platforms such as iCub [9], Cozmo is a fraction of the price; platforms such as the iCub are approx. £300,000 whereas Cozmo is approx. £200. This makes it a low-cost simulation platform as well as a practical platform for teaching.

Figure 2 – Cozmo

Cozmo is versatile and relatively successful on a commercial level which creates a greater support network from the company for the community. For example, Anki have a forum, accessible here [10], where people can post their work as well as ask for support. The Cozmo app contains interfaces for all ages to appeal to the commercial market, as well as a Python API for developers. For this project, Cozmo will be used as a small-scale example platform for testing in realistic environments. Its small size makes it possible to bridge the gap between pure simulation and the real world. Since ROS will be the foundation of the implementation, the architecture developed can be applied to a wide range of other platforms with minor modifications and, therefore, feeds directly into the current research undertaken at Heriot-Watt University.

1.4 Aims and Objectives

Few cognitive architectures address the issue of perceiving the world and effecting it [3], therefore my aim is to design and implement an architecture concentrated around investigative visual perception. Using ROS and Cozmo's Python API, the aim of this project is to successfully employ autonomous mapping, self-localisation and navigation in real-world environments with limited resources, to enable the detection of people and hazards, executing appropriate responses where necessary. The principal objectives to achieve this are as follows:

Objective 1: *Scenarios*
The first objective is to design real-world environments for exploration. These will be used as small-scale replications of environments in which the proposed architecture can be applied for real world situations. The environments should be designed in such a way that they can be modified into different configurations with ease.

Objective 2: *Self-Localisation and Mapping*
The next objective is to investigate state-of-the-art SLAM techniques to be used on Cozmo. The SLAM module must account for the limitations of the hardware, while still producing reliable and accurate results. The main challenge of SLAM in monocular systems is the unknown scale of the map and the unknown estimated trajectory. Therefore, scaling should be attempted, and reality vs abstraction explored. This, along with *Objective 1,* will contribute towards the agents' perception of the world.

Objective 3: *Navigation*
The third objective is to design and implement a visual navigation module based on the map produced from *Objective 2.* The success rate of navigation should be investigated.

Objective 4: *Object Detection*
The fourth objective is to investigate state-of-the-art object recognition and integrate a module into the system. These modules need to allow for detecting markers that can represent different objects. This simulates object detection approaches which can be employed on more sophisticated robots with a wider range of sensors.

Objective 5: *Goal Generation*
The next objective is to design and implement tasks for the agent to execute based on object recognition and navigation. These must be relevant to the scenarios created in *Objective 1.* These should act as a 'proof of concept' where the architecture can be used in real life situations.

Objective 6: *Behaviour Generation*
The final objective is to implement a simple behaviour module. The behaviour generation should be implemented for both goal-based behaviour, stemming from *Objective 5*, and reactive behaviour stemming from environmental situations. The agent should perform the correct behavioural responses, based on the situation presented.

2 Background and Related Work

This section aims to give some context to current state-of-the-art, advances and areas for improvement in this field of research, as well as some similar approaches used as inspiration. This includes a general insight into cognitive architecture, visual perception algorithms, SLAM and navigation techniques.

2.1 Cognitive Architecture

Cognitive architecture combines cognitive science and artificial intelligence to mimic natural cognitive systems. The properties of such architectures revolve around the representation, organisation and utilization of knowledge, which enables constant development and the ability to perform new tasks [4]. Thus, modelling the systems in the human mind aims to create human-level artificial intelligent behaviour in a complex environment. A cognitive architecture can be broken down into the following six key components: perception, attention, action selection, memory, learning and reasoning.

The ability to recognise situations through perception enables the agent to make connections between its environment and its knowledge [3]. Therefore, perception is a fundamental feature of an architecture. Attention closely follows on from perception and is important for deciding where to allocate perceptual resources, enabling relevant information to be gathered from the environment. The main criteria associated with action selection are relevance, utility, internal factors and motivations. The memory of an architecture comes in many forms; for example, sensory memory, long term memory, global memory. This allows for adaptation to environmental changes, and the majority of architectures have some form of memory [4]. Learning, although not fundamental to a cognitive architecture, provides a valuable addition to the system: the capability to improve performance over time. Reasoning brings a more philosophical element to a cognitive architecture; however, reasoning can be spilt into two parts, practical and theoretical. With the target of practical reasoning being to find the 'next best' response to situations and act upon it, theoretical reasoning steers more towards manners such as establishing beliefs.

Kotseruba et al. [4] performed a comprehensive survey of the last 40 years of cognitive architectures, discovering that although most architectures are theoretical, visual input accounts for more than half of all the implemented modalities for perception. Multi-modal perception is widely used in cognitive architectures, as the results prove to be much more accurate.

2.2 Visual Perception Algorithms

Applications for visual algorithms include the automotive industry, for purposes such as autonomous vehicles [11] and collision avoidance [12]. Other applications include deep learning for pose estimation [13]. Few cognitive architectures use deep learning for image processing although it is a popular topic for computer vision research [4].

2.2.1 SLAM

Another interesting area of research for visual perception is SLAM. This can be implemented with the use of cameras, laser-range finders, and other hardware. A popular SLAM technique is GMapping, based on the odometry of the robot and a laser scanner. This is the 'default' for most mobile robots [14]. GMapping can be described as a 'highly efficient' Rao-Blackwellized particle filter for producing grid maps from laser range data [15], [16]. Another technique, PTAM (parallel tracking and mapping) [17], is a keyframe based SLAM for stereo systems (two cameras), however it does not support loop closure due to no loop detection.

SLAM can be utilised on systems with a combination of hardware compositions, to provide multi-modal perception or, for example, with one camera; monocular SLAM. Although monocular SLAM is a popular choice for its simple hardware, it is not without its disadvantages. For example, depth cannot be directly inferred from one camera and without a sense of depth, it is difficult to infer distance. Furthermore, algorithms for monocular SLAM tend to be much more complex than multi-modal SLAM. This has introduced an interesting research topic for visual algorithms.

Wang et al. [18] developed an algorithm for moving object detection (MOD) using monocular vision. This work presents an interesting approach of tackling the shortcomings associated with monocular based visual SLAM. Their proposed algorithm was tested on a hand-held monocular camera and was shown to improve the robustness of both state estimation and the mapping processes of SLAM.

R. Mur-Artal et. al [19] developed ORB-SLAM, based on the ORB algorithm as a feature based SLAM for monocular systems. They implemented a 'survival of the fittest' based strategy to select points and key frames, which the paper suggests as being robust. This was tested on 27 sequences and was proven to produce a trackable map that only extends when the scene content changes. Further to this, *R. Mur-Artal et. al* extended their algorithm to produce ORB-SLAM2 [20] for monocular, stereo and RGB-D cameras. Due to Cozmo having a monocular camera, with no depth sensor, these successes in overcoming the constraints of monocular SLAM motivate my approach of implementing SLAM onto Cozmo.

2.2.2 Feature Detection

To use visual inputs for self-localisation and mapping, features must be extracted from the visual source. SIFT [21], SURF [22] and ORB [23] continue to be popular algorithms for feature detection and are actively used for SLAM. This section will outline how each of these algorithms work.

SIFT [21] extracts key points (KP) and computes its descriptors. First, scale-space extrema detection is computed. This is implemented using a difference of Gaussian function, which approximates the Laplacian of Gaussian. This identifies potential KP's, that are invariant to scale and orientation. Each KP location is then refined to get more accurate results via KP localisation. Location and scale are considered, and if below a certain threshold, the KP is discarded. Next, an orientation is assigned to each KP based on local image gradient directions to achieve invariance to image orientation, scale and location. A KP descriptor is then assigned, by measuring the local image gradient around each KP and then transforming this into a vector representation to allow for distortion and change in illumination.

SURF [22] approximates the Laplacian of Gaussian using Box Filter. The advantage of this is that convolution with Box Filter can be easily calculated with the help of integral images. SURF relies on the determinant of the Hessian matrix for both scale and location. Orientation is assigned using Haar-wavelet responses of the neighbourhood of the KP. Gaussian weights are then applied, and these are plotted. The dominant orientation can then be estimated but calculating the sum of all responses within an orientation window. The descriptor is formed using wavelet responses around each KP neighbourhood. These are then divided into sub-regions where wavelet responses are taken and are combined to form a vector.

An advantage of SURF is that it is 3x faster than SIFT, and it is good at handling images that contain blurring and rotation. The disadvantage is that it is not good at handling viewpoint or illumination changes. Therefore, this type of feature algorithm would not be suitable for use on Cozmo; Cozmo's viewpoint of a feature will constantly be changing as it navigates the environment.

ORB [23] was developed as an alternative to SIFT and SURF. ORB combines FAST [24], [25] to find KP's and BRIEF [26] for the descriptors. Once the KP's are found, a Harris corner measure is applied to find the top points. However, FAST does not compute orientation, therefore the intensity weighted centroid is computed, with the located corner at the centre. The direction of the vector from the corner point to the centroid gives the orientation. The descriptor is formed using BRIEF. ORB 'steers' BRIEF according to the orientation of the KP, because BRIEF is known to perform badly with rotation. A rotation matrix is calculated and applied to the feature set. To overcome the downfalls of BRIEF, ORB runs a greedy search to find the KP's that are uncorrelated, have high variance and a mean close to 0.5.

ORB feature detection was chosen by the designers of ORBSLAM, so the system can use the same features for mapping and tracking, as well as place recognition to perform frame-rate re-localisation and loop detection [19].

2.3 Navigation

Navigation for mobile robots is usually divided into visual and lidar-based navigation where currently, visual navigation is gaining popularity due to the advent of autonomous vehicles. Approaches such as experience-based robust localisation [27], represents the state-of-the-art in this field. ORBSLAM, mentioned earlier, is another approach of tackling this problem. The basic principles of generating velocities to drive the robot's wheels, however, have not changed since the early years of robotics; Generate an angular velocity that aligns the front of the robot with its goal and a linear velocity that reduces the distance to the goal [28]. Visual and topological navigation are two approaches used in this project due to the lack of a laser sensor.

2.3.1 Topological navigation

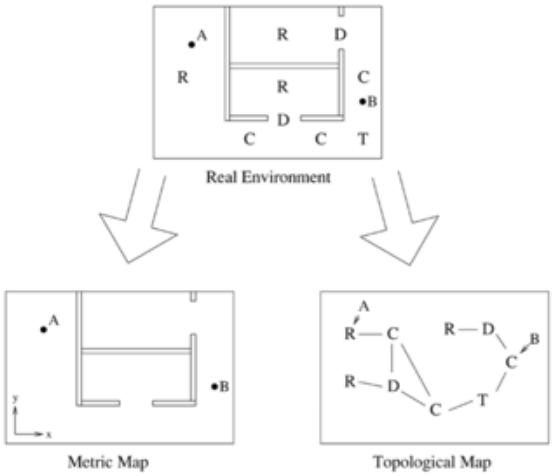


Figure 3 - Metric and topological maps. Image source [29]

The most common approach to navigation is using a grid map [30] such as in the ROS navigation stack [31]. These are then used to navigate to a given goal in the environment using Dijkstra's algorithm or A* for global path finding, and the dynamic window approach for local path planning and dynamic obstacle avoidance [28]. However, navigating more complex environments and/or longer routes can be a challenge for grid-based navigation due to the size of the search space. To counteract and additionally be able to associate high-level behaviours with certain locations in the environment, topological navigation is used, which can either be hand-crafted or automatically learned, e.g. [32]. This type of navigation reduces the search space to a simple graph search in a much-reduced search space (see Figure 3). Grid map-based navigation is then used to navigate from one topological node or 'waypoint' to another, traversing the path to the final goal. Recent approaches aim to modernise this type of navigation by making it robust to changes in the environment [33] which shows that it is still state-of-the-art in robotic navigation.

2.3.2 Visual navigation

One of the earlier approaches to visual navigation [34] uses an omni-directional camera to take pictures at certain locations, i.e. nodes in a topological map, and in-between these nodes. During the navigation process, the current view of the robot is matched to this image database and the robot is moved in the direction that decreases the difference between the images. This type of feature mapping is thought to be similar to insect vision and navigation, which inspired other approaches to visual robotic navigation such as in [35]. Other examples for visual navigation approaches include, but are not limited to, using video streams and gradient orientation histograms [36], on-line reconstruction and recognition of 3D objects [37], and identification and localisation of landmarks in images [38], etc.

Apart from outdoor navigation for autonomous vehicles mentioned above, these types of systems have in the past been popular for indoor applications such as museum tour guide robots, e.g. [39], [40]. However, while visual navigation is less frequently used for wheeled indoor robots nowadays, it is often the only form of navigation for Unmanned Aerial Vehicles (UAVs) such as [41], [42] which has led to the development of approaches such as ORBSLAM, mentioned above.

3 Project Plan

3.1 Gantt Chart

An overview of the milestones for this project can be seen in the Gantt chart below. The objectives outlined in Chapter 1 correspond to the main milestones for completion, more information on these can be found in section 1.4. Section 3.3 outlines the structure of this report and which chapters align to the objectives.

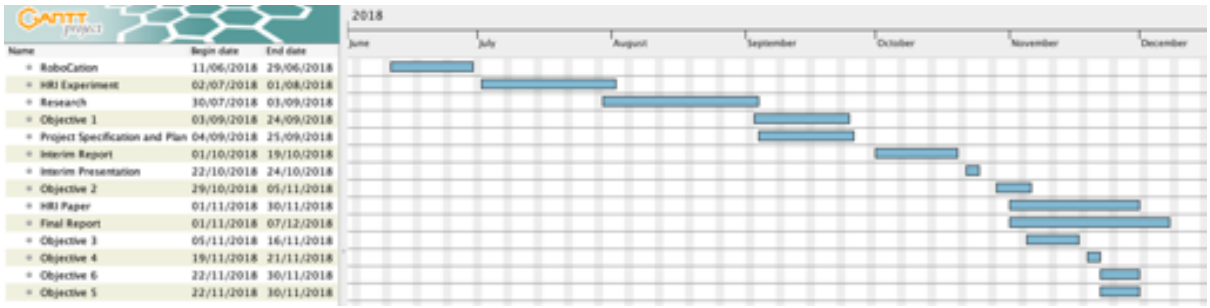


Figure 4 - Gantt Chart

3.2 Environment Scenarios

Addressing *Objective 1*, two real-world inspired environments have been constructed for the purpose of this project. The first environment is built with Lego to create a town with roads, junctions and buildings. This will further be referred to as the 'Lego town'. The decision to use Lego was based on the dimensions of Cozmo. This provides the opportunity to imitate every-day occurrences in which autonomous system support could be useful. Small objects will be placed around the environment to simulate realistic obstacles. Locations in the environment will be allocated to buildings such as: hospital, fire station etc. Figure 4 shows a bird's eye view of a Lego town configuration, while figure 5 shows an example of an object in the environment.

Figure 5 - Lego town configurations

Figure 6 - Object in the Lego town

The second environment is a replication of an oil rig, to be used in connection with the ORCA hub research project [1]. This is constructed from wood, cardboard and plastic plumbing pipes. The 'oil rig' has 3 levels suitable for exploration. Figure 6 shows the constructed oil rig from two different perspectives. Ramps will be assembled between the three levels, so the agent can manoeuvre between them. The Lego town will be the main environment used through this project, with the aim of applying the successful system to the more complex oil rig environment if time permits.

Figure 7 - Oil rig

Both environments have been assembled in a way that they can be modified to create new states and problems. For example, the interior 'walls' of the oil rig will be made of cardboard and will be easily removed and modified to create different size and shaped rooms. The roads of the 'Lego town' are made up of jigsaw-like segments, that can be interchanged into different configurations. Each environment will contain relevant obstacles and situations for the agent to navigate around, of which examples have been shown in the figures above. These environments were specifically chosen as they relate to practical situations where these types of systems could be useful.

3.3 Proposed Architecture

Figure 7 represents the initial design of the proposed architecture. Cozmo has its own local Wi-Fi network, which is used to connect to a tablet. The connection is made through the Cozmo app which is installed on the tablet. The tablet is then connected via a USB cable to a laptop and the 'SDK mode' is selected on the app. This setup is required for any development and testing on Cozmo. An already existing ROS Cozmo driver can then be executed to publish information from Cozmo for use and to control its wheels and actuators via commands sent to the driver.

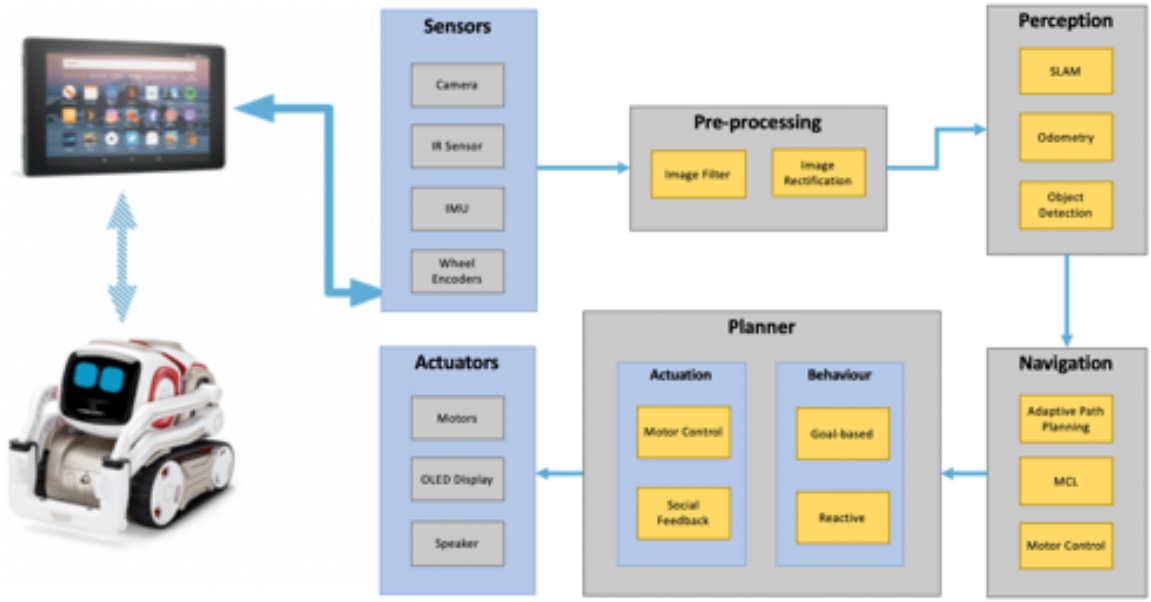


Figure 8 - Proposed architecture

The pre-processing module will contain the necessary steps required for any form of computer vision; the raw input image from the sensors will be processed via filtering and rectification. Image pre-processing will be described in Chapter 4. The perception module will consist of the necessary components to form a 'world model' for the agent. All information gathered in this module will be what the agent uses to perceive the world. This includes SLAM, object detection, and odometry, contributing to *Objective 2* and *Objective 4.* Object recognition is outlined in Chapter 5, while SLAM is described in Chapter 6. The navigation module uses perception as an input, allowing for exploration and execution of tasks. This corresponds to *Objective 3* and will be described in Chapter 7. The planner module, covered in Chapter 8, consists of both behaviour and actuation and is the main source of feedback. This is where the previous modules will be combined to produce a complete system, capable of executing tasks and behaviour. This chapter corresponds to *Objective 5* and *Objective 6.*

4 Pre-processing

4.1 Camera Calibration

When it comes to computer vision, the quality of the images is paramount. Even two identical cameras will never produce the same image due to slight variations during the manufacturing process of the lenses and sensors. Therefore, camera calibration is fundamental in any situation where vision is the main supplier of information; this ensures accuracy and reliability by removing distortions introduced by the recording hardware. Therefore, before any form of SLAM can be used, it is important to calibrate the camera on Cozmo. By calibrating the camera, intrinsic and extrinsic parameters can be derived along with distortion coefficients.

Radial distortion can be identified where straight lines appear curved, and can be solved by the following equations:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Tangential distortion can be identified where some areas of the image look nearer than expected, which is caused by the camera lens not being parallel to the imaging plane. This can be solved with the following equations:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{corrected} = y + [2p_2 xy + p_1(r^2 + 2y^2)]$$

These can be rectified by calculating the 5 following distortion coefficients:

$$Distortion\ coefficients = (k_1, k_2, p_1, p_2, k_3)$$

Intrinsic parameters contribute to what is referred to as the camera matrix. These parameters are specific to the camera being used and the matrix contains the focal length (f_x, f_y) and the optical centre (c_x, c_y) of the camera. The camera matrix is shown below:

$$Camera\ matrix = \begin{matrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{matrix}$$

4.1.1 Implementation

ROS has a pre-made driver for the calibration of a monocular camera, which is applied here. The method carried out, as described in the ROS documentation [43] will calculate the height and width of the image, the distortion coefficients, the camera matrix, the rectification matrix and the projection matrix.

To calibrate the camera, a 'checkerboard' with known dimensions is printed. The one used here is printed onto A4 paper and contains squares with 108x108mm dimensions, equating to 9 squares by 7. This allows for the camera to use an 8x6 grid, due to the calibration using the interior vertex points of the checkerboard [43].

While *roscore* and the Cozmo driver are running, the calibration node can be executed with the following command. This loads the image topic to be calibrated and runs the node:

> *"rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/cozmo_camera/image camera:=/cozmo_camera"*

The documentation recommends the following movements are completed for optimal calibration:

- Place checkerboard to the camera's left, right, top and bottom of field of view
- Move checkerboard from left to right
- Move checkerboard from top to bottom
- Move checkerboard toward and away from camera
- Checkerboard filling the whole field of view
- Checkerboard tilted to the left, right, top and bottom

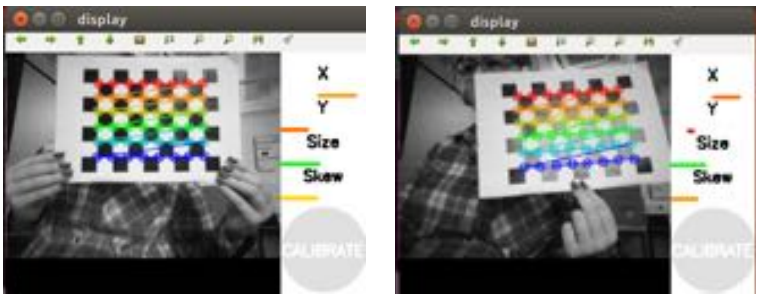


Figure 9 - Camera calibration

The checkerboard is moved around the field of view until the 'calibrate' button lights up. This will happen when enough samples have been recorded. Once the 'calibrate' button is pressed, the driver does the appropriate calculations based on the samples and the results are printed directly into the terminal.

4.1.2 Results

The calibration process was carried out three times and the average values were calculated. The average calibration values will be used throughout the project. The results of each calibration can be found in the appendix.

The values to be used are as follows:

Table 1 - Camera calibration results

Image Width	320	
Image Height	240	
Camera Matrix	$\begin{matrix} 311.534 & 0 & 171.893 \\ 0 & 312.647 & 128.119 \\ 0 & 0 & 1 \end{matrix}$	
Distortion Coefficients	k_1	0.166826
	k_2	−0.000966
	k_3	0.000000
	p_1	0.010706
	p_2	−0.001481
Rectification Matrix	$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$	
Projection Matrix	$\begin{matrix} 331.137 & 0 & 170.868 & 0 \\ 0 & 333.491 & 129.698 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$	

4.2 Image Filtering

Apart from distortions, noise is another factor that influences the quality of computer vision algorithms. Hence, image filtering and enhancement is fundamental for computer vision. There are many types of image filters that can be used, depending on the desired result. Image enhancement is widely used for many applications, particularly for processing data for autonomous machine perception. Here, the aim is to reduce noise for feature extraction therefore image blurring and smoothing are the most appropriate.

Linear filters are most common for smoothing images. Gaussian blurring is a low pass filter, often used for pre-processing to enhance image structures. The Gaussian function in two dimensions can be written as:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

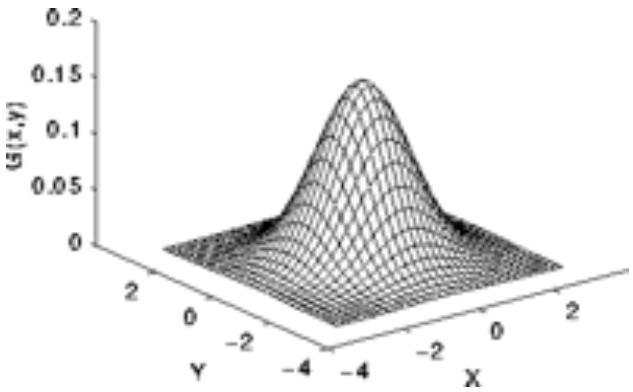


Figure 10 – 3D Gaussian function

Since the resulting Gaussian bell-curve above is 3-dimensional, it has to be projected onto a 2-dimensional plane in order to work as an image filter. Furthermore, a Gaussian function is continuous, but pixel values in images are discrete, therefore, the z values are discretised. The result for a 5x5 Gaussian filter can be seen below. The numbers in the squares are then normalised so that their values sum up to 1. For the filtering process, the centre point of the resulting filter is placed over each pixel in the original image, the values of all the pixels underneath the filter are multiplied by the corresponding filter value, and summed. The resulting number is then used as the value of the pixel in the new images that corresponds to the location of the centre of the filter in the original images. Through this process, the original image is blurred which removes the most common forms of noise in images. To increase the speed of this operation, most state-of-the-art image processing libraries use convolution.

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Figure 11 – 2D discretised Gaussian function

The goal is to reduce noise within the image, without removing too much detail, so that key points can still be detected and used for SLAM which motivates the use of a bilateral filter.

Like the Gaussian filter, bilateral filters use a filter mask where the pixel values under the mask are multiplied with the normalised filter values, summed, and finally used to represent the value of the pixel in the new image that corresponds to the centre pixel under the filter in the original image. The filter values, y, in position i, are calculated using the following formula:

$$y_i = f_r(\|I(x_i) - I(x)\|)g_s(\|x_i - x\|)$$

These are then summed and normalised to create the pixel value for the new image:

$$I^{filtered}(x) = \frac{1}{\sum_{i\in\Omega} y_i}\sum_{i\in\Omega} I(x)y_i$$

Where I is the original image, $I^{filtered}$ is the new image, x is the pixel position, Ω is the size of the filter, $f_r$is the range kernel for smoothing differences in intensities which can be Gaussian, and g_s is the spatial kernel for smoothing differences in ranges which can also be Gaussian. This makes the bilateral filter a noise reducing and edge preserving non-linear filter that is not just based on the Euclidean distance, but also on radiometric differences such as colour intensity.

4.2.1 Implementation

The image is subscribed from */cozmo_camera/image* as the input image source. A Gaussian blur filter is then applied to smooth the image and reduce noise. Gaussian filtering convolves each input point in the input array with a Gaussian kernel, and then sums these values, to produce an output array. The OpenCV function that applies this filter is:

"blur = cv2.GaussianBlur(src, kernelSize, sigmaX[dst[sigmaY[borderType]]])"

Where: *src = input image*
kernelSize = gaussian kernel size
SigmaX and sigmaY represent the standard deviation in the x and y directions
dst = output image
borderType = pixel extrapolation method

The more an image is smoothed, the fewer edges are detected. Although the objective of this section is to smooth the images, the edges need to remain distinct to avoid any interference with the performance of the feature extraction element of SLAM. Therefore, to counteract smoothing of the edges, a bilateral filter is applied to the image to retain the edge definition.

The bilateral filter is a combination of domain and range filtering. The OpenCV function for the bilateral filter is:

"blur = cv2.bilateralFilter(src, d, sigmaColour, sigmaSpace)"

Where: *src = input image*
d = diameter of pixel neighbourhood
sigmaColour = filter sigma in the colour space
sigmaSpace = filter sigma in the co-ordinate space

Once the filters are applied, the image is published to */cozmo/image_filter.*

4.2.2 Results

First the Gaussian filter was applied on its own to the raw image. The images below show the unfiltered frame on the left, and the filtered frame on the right with a kernel size of 3x3. As seen below, the key effect of the Gaussian filter is a general all-over blurring of the image. Noise is reduced; however, edges are also blurred, which could negatively influence the feature matching algorithm.

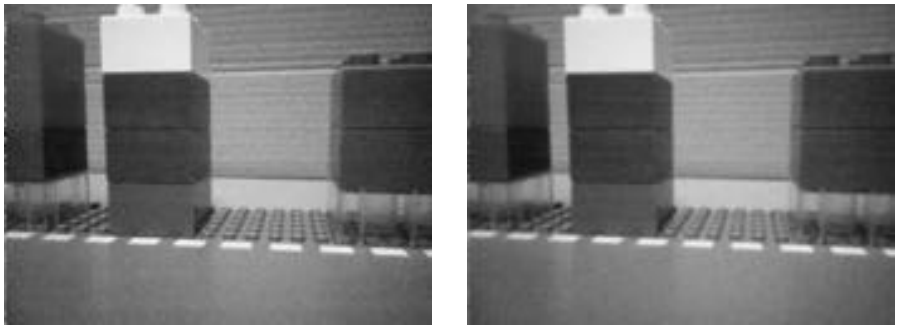


Figure 12 - Gaussian filter

Next the bilateral filter was applied on its own. The parameters shown below are (5,75,75) and, once again, the image on the left is the raw frame, and on the right is the filtered frame. The clear effect of the bilateral filter is the smoothing of the image, but the edges remain distinct. For instance, the contrasting colours are more defined, but the areas with a very uniform colour appear to be smoothed out.

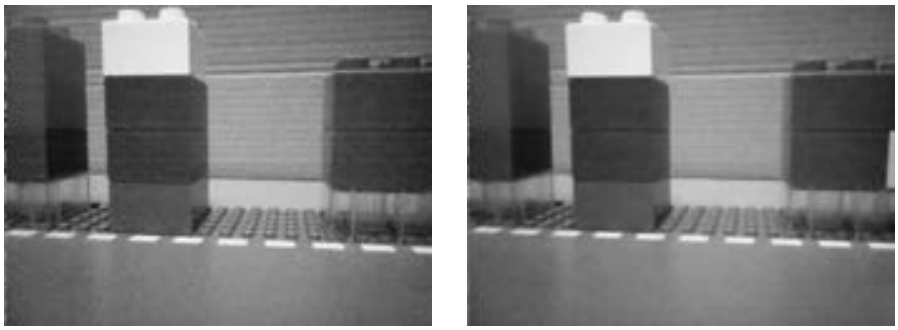


Figure 13 - Bilateral filter

Finally, both filters were applied concurrently. The results are shown in the figure below with the raw frame on the left and the filtered frame on the right. This combination of filters has the most deviation from the original image. This is expected; however, the blurring is too intense, and prevents the feature detection from working altogether.

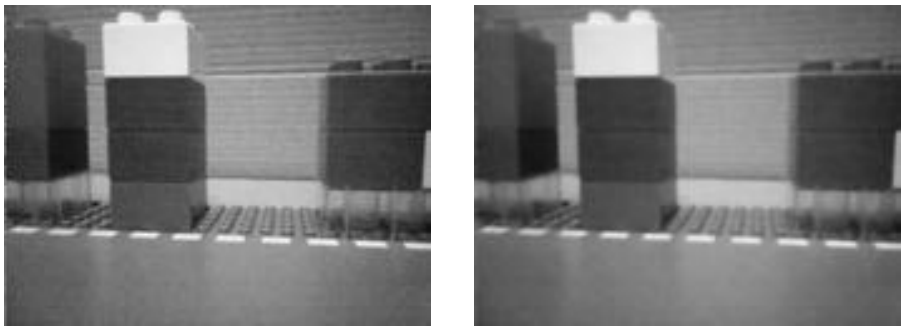


Figure 14 - Gaussian filter and bilateral filter

5 Object Recognition

AR is officially described as “Augmented reality, in computer programming, a process of combining or ‘augmenting’ video or photographic displays by overlaying the images with useful computer-generated data” [44]. AR markers, more generally known as *fiducial* markers, are 2D codes that are detectable via simple template matching. These are used here as simulation of object detection on more powerful hardware (i.e. with better sensors, GPUs, etc.). The markers are generated using the *ARToolKit* [45] and printed at a size of 2cm x 2cm. An example of AR markers can be seen in the figure below.

Figure 15 – Example AR Markers

AR markers will be used to fulfil a proof of concept for the object detection portion of the proposed architecture, addressing *Objective 4*. The implementation of which is described below.

5.1 Implementation

First, the image is rectified via calibration and image processing. This is then the image topic that is subscribed to. The calibration values used are the ones calculated in Chapter 4, which are published by the Cozmo driver. The generated markers are loaded into the program as the ‘object data’, as well as their associated bit files. The rectified image topic is subscribed to, and the markers are detected in each frame. When a marker is detected, it is compared to the loaded object data. If a match is found, a confidence level is calculated. The translation vector and rotation matrix of each marker, compared to the origin of the camera tf frame, is then computed before being converted into a tf frame itself. The detected marker and the transform between the camera and marker are then published.

While x and y can be calculated directly from the image information, the z-value, i.e. the distance to the camera, is based on the size of the actual marker and its perceived size in the image. To this end, the algorithm was provided with the size of the printed marker in millimetres. However, when testing the node, the distance calculated was too large. In order to rectify this, the marker was placed at a fixed distance to the camera, and the size of the parameter was changed so the calculated distance corresponded to the actual distance. The marker was placed at 10cm away from the camera position, which can be seen in the figure

below. The value that correctly calculated distances for the used implementation of AR marker detection and Cozmo's camera was 17mm instead of the actual size of 20mm.

The image below shows the physical set-up to generate ground truth value on the left and the resulting detections on the right. The two grey scale images represent the raw camera image (top) and the detected markers highlighted via a circle (bottom). The right half of the right image shows the cozmo_camera frame and the resulting AR marker frame.

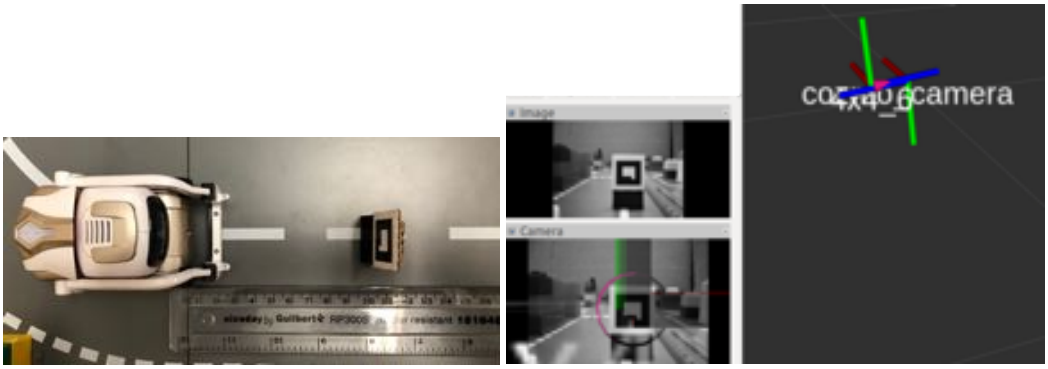


Figure 16 – Scaling the AR markers

As can be seen below, this value was verified using multiple markers at 10cm and 20 cm distances from the camera. Additionally, the marker at 20cm was placed at an angle, which can be seen in the right picture, that shows the left most tf frame at the same angle as the marker.

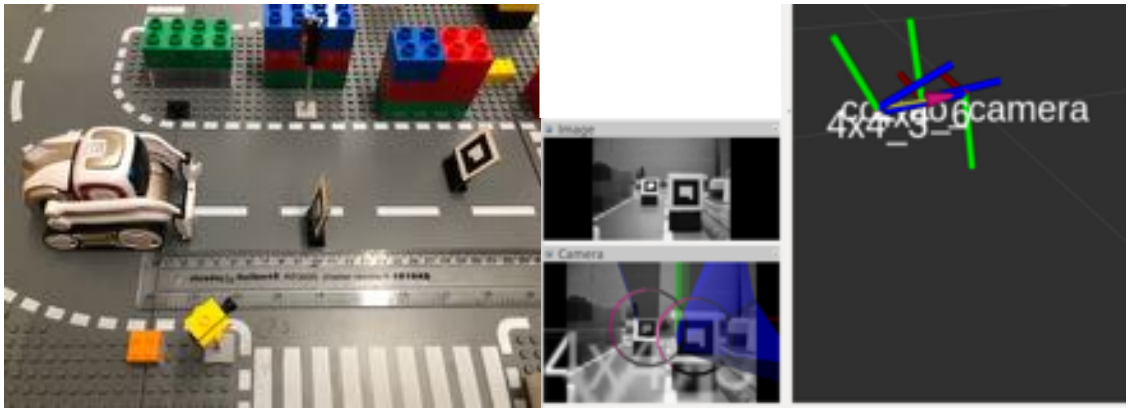


Figure 17 - AR marker verification

A comparative study of AR systems, including the efficiency, accuracy and reliability of such techniques can be found here [46].

6 Self-Localisation and Mapping

In order for any robot to be able to perform tasks in hazardous environments, it first has to reach its location. To accomplish this, it needs to know where it currently is and where it has to be. The most basic localisation method for wheeled robots is odometry. Odometry uses wheel encoders to count how often a wheel rotates. Given the radius of the wheel, the current position of the robot compared to its starting position can be estimated. This process, however, is subject to errors due to wheel slippage. This error is propagated from one time-step to the next and, therefore, continuously increases. Odometry is especially bad on tracked vehicles due to slippage when turning. Therefore, the only viable sensor for verification of location is the camera. A popular state-of-the-art localisation method for monocular cameras is ORBSLAM, which uses ORB features (see Background) and has been adopted for this project to address *Objective 2*.

Before running ORBSLAM on Cozmo, it was first tested it on the Kitti Dataset. The images from this will be used to help outline the process of ORBSLAM. ORBSLAM is an "out-of-the-box" [20] state-of-the-art for SLAM, especially for monocular systems with a limited field of view. ORBSLAM can be used for monocular, stereo and RGB-D camera systems. The following describes the ORBSLAM system developed by *Raul Mur-Artal* et. al [19].

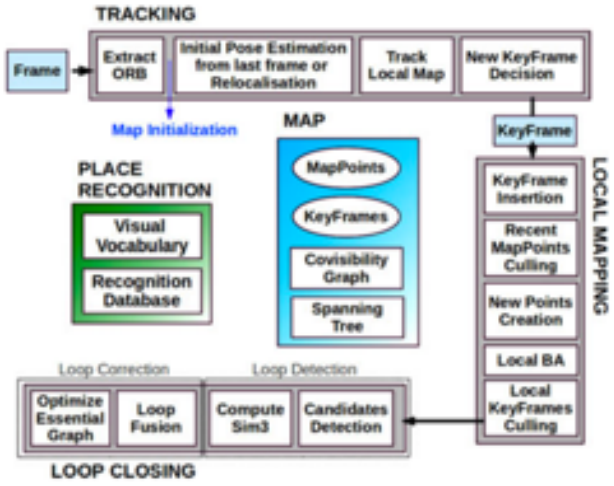


Figure 18 - ORBSLAM System Overview

First, the map is initialised. This process computes the relative pose between two frames, triangulating an initial set of map points. A map point contains the 3D position, viewing direction, ORB descriptor, and the min and max distances which the point can be observed. The ORB features are extracted from the current frame and compared to the reference frame for any matches. A full Bundle Adjustment (BA) (all points and keyframes are optimised except the first keyframe, which remains fixed at the origin) is applied, refining the construction of the map. Bundle Adjustment provides estimates of camera localisations, and

can be defined as “the problem refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter (camera pose and/or calibration) estimates” [47].

The main process of ORBSLAM is made up of three modules running in parallel: tracking, local mapping and loop closing. An outline of these modules is described below.

The task of the tracking module is to localise the camera with every frame and decide when it is appropriate to insert a new keyframe. The ORB features are extracted and compared to the previous frame. A keyframe contains the camera pose, camera intrinsics and the ORB features extracted in the frame.

Figure 19 - ORB feature detection on kitti dataset

If tracking is successful for the previous frame, the initial camera pose estimation is computed using motion-only BA (all points are fixed, only the camera pose is optimised). If the tracking is lost, global re-localisation is required. The recognition database, in the ‘place recognition’ module, is queried for keyframe candidates and these are used for global re-localisation. If a suitable camera pose keyframe is found, the pose is optimised and the candidate keyframe is searched for more map points. The camera pose is optimised again, and the tracking can continue.

Once initial features and an initial pose is computed, it is possible to construct a local map. The map contains: a set of keyframes that share map points with the current frame, a set of keyframes that share neighbours with the other set of keyframes, and a reference keyframe which shares most points with the current frame. Finally, the tracking module decides whether the current frame should be a new keyframe. This will occur if all four of following conditions are met:

1. > 20 frames have passed since the last global re-localisation
2. > 20 frames have passed since the last keyframe insertion
3. Current frame tracks > 50 points
4. Current frame tracks < 90% of points than the reference keyframe

Local mapping processes new keyframes and performs local BA for optimal reconstruction in the surroundings of the camera pose. The local mapping module inserts the prospective keyframes into the map by adding a new node to the co-visibility graph and updates the corresponding edges. A co-visibility graph is a representation that captures which features

are observed at the same time [48]. The image below shows a map M_k, at a given time k; an undirected graph of 'landmarks' with edges that signify co-visibility [49].

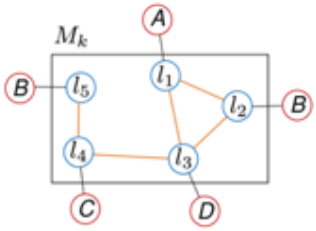


Figure 20 - Co-visibility graph

The spanning tree is also updated to link the points in common between the new keyframe and the already existing keyframes. A spanning tree is a subset of the graph, which has all the vertices covered with the minimum possible number of edges. Map points then get 'culled' to confirm that the only map points saved are trackable. The two conditions must be met for a map point to avoid being culled:

1. The tracking module must find the point in > 25% of the frames in which it is perceived to be visible
2. The map point must be able to be observed for at least three keyframes since the creation of the map point

This ensures only high-quality points are retained, and noise is subsequently decreased. New map points are created by triangulating the ORB features between keyframe connections in the co-visibility graph. Local BA is then performed (all points in the local area are optimized while a subset of keyframes is fixed). The final task for the local mapping module is culling necessary local keyframes. If 90% of the map points belonging to a keyframe have been visible in at least three other keyframes, then the keyframe is culled.

Loop closing takes the last keyframe, and searches for loops with every new keyframe [50]. If a loop is detected, a similarity transformation is computed from the current keyframe to the loop keyframe. This is required to monitor the error due to there being seven degrees of freedom in which the map can drift (three rotations, three translations and a scale factor). The figure below shows the map before loop closure.

Figure 21 - Before loop closure

If a similarity is discovered, it is optimised. A guided search is then performed, and it is optimised again. If the loop is accepted, it is then aligned, and the duplicate map points are fused. The co-visibility map is updated, and the transforms are concatenated, to align both sides of the loop. The final step is to apply a pose graph optimisation to distribute the loop closing error and to achieve global consistency. The figure below shows the map after loop closure.

Figure 22 - After loop closure

The final map produced on the Kitti dataset when replicating the results of the original paper can be seen below.

Figure 23 - Complete map using kitti dataset

In summary, each of the black dots or 'map points' in above map contains the 3D position, viewing direction, ORB descriptor and the min and max distances at which the point can be observed. All the keyframes along the path contain the camera pose, camera intrinsics and the ORB features extracted in the frame. Additionally, as can be seen from Figure 20 and the misalignment of the red area, ORBSLAM is susceptible to scale drift which is somewhat corrected via scale drift aware loop closing [50] (seen in Figure 21) to produce the final map in Figure 22.

6.1 Implementation

The source code for ORBSLAM was forked from the creators of the system. This can be viewed via GitHub at https://github.com/hmt2/ORB_SLAM2. ORBSLAM requires a settings file as a parameter, which contains the camera parameters, ORB parameters and the viewer parameters. A new settings file was created with the calibration values recorded in Chapter 4 of this report, specific to Cozmo.

6.1.1 Dataset Construction

Datasets were recorded for consistency during testing. These were recorded using *rosbag*, to record the relevant topics and their information. Cozmo was teleoperated around the Lego world, whilst ORBSLAM was running. The steps for recording a map are as follows:

Method: Dataset Construction

- Run roscore:
  "roscore"
- Run the ROS Cozmo driver:
  "rosrun cozmo_driver cozmo_driver.py"
- Run the ROS teleoperation node:
  "rosrun cozmo_driver teleop_key.py _lin_vel:=0.03 _ang_vel:=0.3"
- To record the topics:
  "rosbag record topic_names"
- Run the ORBSLAM node:

"cd <ORBSLAM_DIR>; rosrun ORB_SLAM2 Mono Vocabulary/ORBvoc.bin Examples/Monocular/Cozmo.yaml /cozmo_camera/image 1"

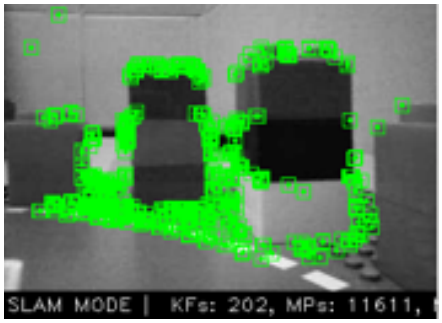


Figure 24 - ORB feature detection

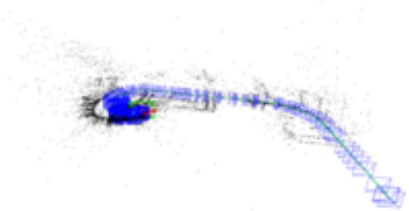


Figure 25 - Map before loop closure

During mapping, the robot started in the bottom right of figure 24. As can be seen above, the distance estimation changed significantly after the second turn. However, the loop-closure rectified this mistake as can be seen in the figure below. Nevertheless, the resulting scale does still not reflect the true size of the environment. This will be addressed later.

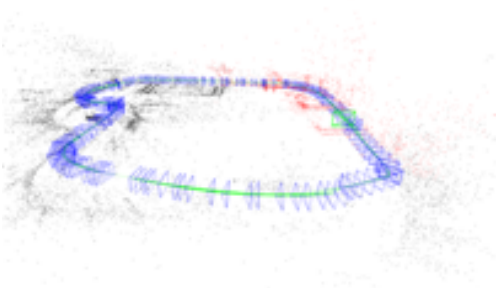


Figure 26 - Map after loop closure

6.1.2 Saving the Map

While ORBSLAM provides a localisation mode where the camera is only localised in the map without updating it, in its original implementation, this map cannot be saved and reloaded. To be able to use the maps recorded in the dataset, ORBSLAM was modified to save the map and reload it when necessary [51]. To do this, an extra parameter is added to the command to run ORBSLAM. This can be set as either 1 or 0 depending on whether the map should be loaded or not. To save the map, the parameter is set to 1 (i.e. true). In addition, the location and name of the map are set in the config file given to ORBSLAM on start-up. This will save the map as a binary containing all created point-clouds of features when ORBSLAM is closed down via Ctrl+C. If the value is set to 1, ORBSLAM will automatically load the map saved as map.bin and overwrite it.

6.1.3 Current Position of Robot

To successfully implement navigation based on the constructed map, the current position of the robot is required. To this end, the implementation discussed here was used: https://github.com/raulmur/ORB_SLAM2/pull/102 which creates a tf tree that connects the map frame to the camera_pose frame, where the map represents the location of the first

recorded key frame during mapping, and camera_pose is the current position of the camera according to ORBSLAM. In addition to that, the "Mono" node was later modified to publish a topic containing the current position of the robot for the localisation node to subscribe to. The camera_pose frame is calculated from the ORBSLAM camera_pose. This is completed using the Hamilton product of two quarternions.

For two quaternions $a_1 + b_1 i + c_1 j + d_1 k$ and $a_2 + b_2 i + c_2 j + d_2 k$, the Hamilton product can be calculated as:

$$HamiltonProd = (a_1 + b_1 i + c_1 j + d_1 k)(a_2 + b_2 i + c_2 j + d_2 k)$$

$$= (a_1 a_2 + a_1 b_2 i + a_1 c_2 j + a_1 d_2 k) + (b_1 a_2 i + b_1 b_2 i^2 + b_1 c_2 ij + b_1 d_2 ik)$$
$$+ (c_1 a_2 j + c_1 b_2 ji + c_1 c_2 j^2 + c_1 d_2 jk + (d_1 a_2 k + d_1 b_2 ki + d_1 c_2 kj + d_1 d_2 k^2)$$

$$= (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2)i$$
$$+ (a_1 c_2 - b_1 d_2 + c_1 a_2 - d_1 b_2)j + (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2)k$$

A 3x3 matrix is formed to store the current camera pose. From this, the rotation of the current pose can be extracted and is saved as a quaternion, *tfqt*.

To calculate the camera translation, Tc, the origin is set as the pose at: $T = \begin{bmatrix}(0,3)\\(1,3)\\(2,3)\end{bmatrix}$

This is multiplied by a rotation matrix to get the right co-ordinate frame (i.e. rotation by 270 degrees in XZ):

$$Tc = T \times \begin{bmatrix}0 & 1 & 0\\0 & 0 & 1\\-1 & 0 & 0\end{bmatrix} = \begin{bmatrix}Tc(0)\\Tc(1)\\Tc(2)\end{bmatrix}$$

To calculate the translation into the world frame:

$$H1 = \begin{bmatrix}tfqt(3)\\tfqt(0)\\tfqt(1)\\tfqt(2)\end{bmatrix}$$

$$H2 = \begin{bmatrix}tfqt(3)\\-tfqt(0)\\-tfqt(1)\\-tfqt(2)\end{bmatrix}$$

$$Ht = \begin{bmatrix} 0 \\ Tc(0) \\ Tc(1) \\ Tc(2) \end{bmatrix}$$

The translation is then calculated as:

$$TSQ = HamiltonProd[HamiltonProd(H1, Ht), H2]$$

$$Translation\ T = \begin{bmatrix} TSQ(1) \\ TSQ(2) \\ TSQ(3) \end{bmatrix}$$

The camera_pose tf tree in the correct co-ordinate frame is saved as:

$$Position\ x = T[0]$$
$$Position\ y = T[1]$$
$$Position\ z = T[2]$$

$$Orientation\ x = tfqt[0]$$
$$Orientation\ y = tfqt[1]$$
$$Orientation\ z = tfqt[2]$$
$$Orientation\ w = tfqt[3]$$

Once the tf tree is created, it needs to be connected to the tf tree of the robot itself. To follow convention, the odometry frame of the robot (which is where the robot is turned on) is connected to the map frame. In order to calculate the translation and rotation, the position of the camera_pose (Cp) in relation to the map and the position of the camera_link on the robot in relation to the odometry (Cl) are calculated. In order to get the rotation difference between map and odometry, the rotation matrix of Cp is multiplied with the inverse rotation matrix of Cl. The resulting rotation matrix is then multiplied with the translation vector of Cl to ensure that both Cp and Cl are rotated accordingly and is then subtracted from the translation vector of Cp. The resulting translation and rotation then form the connection between the odometry and map. This is done automatically via the transform_publisher.py script in the waypoint_nav package which implements a static transform publisher. This also offers a service, to trigger recalibration in case the robot has been picked up or the odometry error becomes too large.

6.1.4 Scaling

As hinted on above, the main shortcomings of monocular SLAM are the lack of scaling and the unknown estimation of trajectory. Therefore, for accurate navigation and self-localisation, it is necessary to produce some form of scaling.

The difference between the actual distance travelled and the change in z value in the image plane for the translation can be calculated. The relationship between these two values can be referred to as the scaling factor. This aligns Cozmo's perception of the world with the actual size of the world.

Figure 27 - Cozmo at Position A

When Cozmo is position A, the front edge of Cozmo is aligned with the 0cm mark on the ruler. When Cozmo travels to the 10cm mark, position B, the change in z translation value is recorded and therefore, a scaling factor of 0.642 can be applied to the translation for the ground truth. This is not perfect because the scale of the map might be inconsistent and, therefore, change from one part of the map to another but it was sufficient for the conducted experiments. This will be discussed further in the Future Work section.

Figure 28 - Cozmo at Position B

6.2 Results

Mapping the environment presents an extensive and time-consuming process, due to several problems, such as: scaling, errors with loop closure, losing localisation etc. Some of these attempts are shown below.

Below are some examples of successful maps:

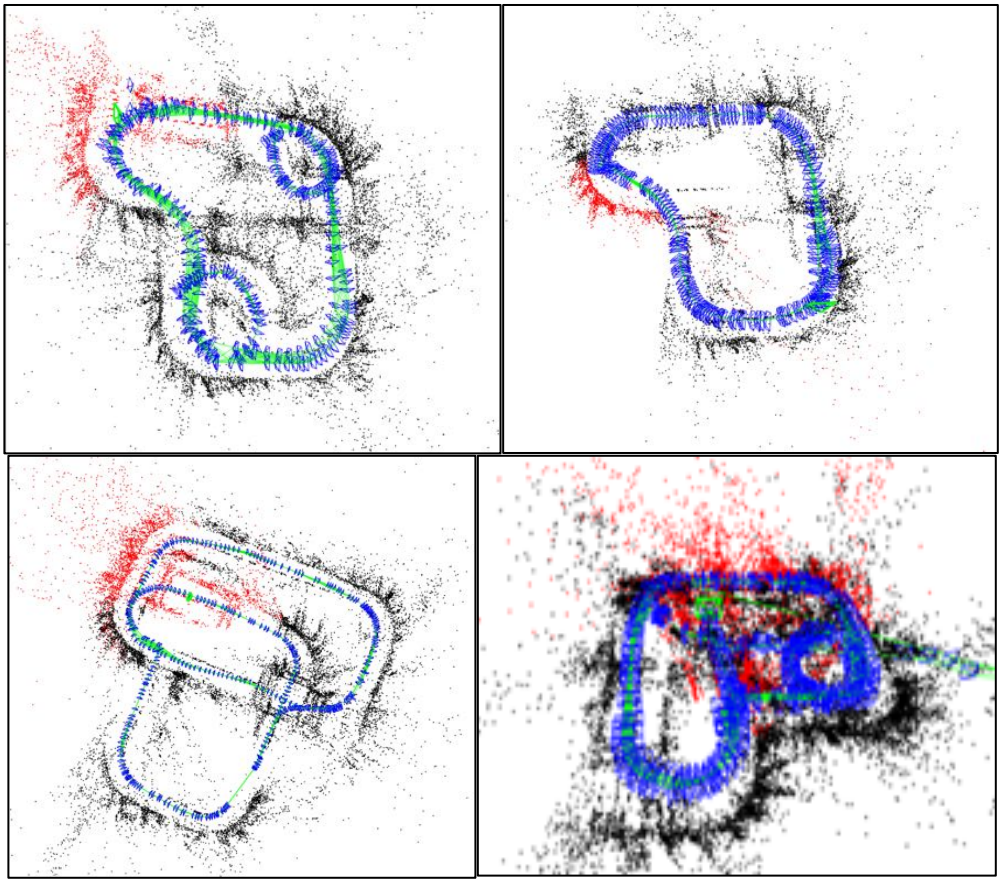


Figure 29 - Successful maps

As seen in the figures above, these maps were successful. Each map gives a clear depiction of the environment and the loop closure was achieved. They show clear paths in which the robot took. However, note that the successful maps are simple shapes without too complex routes.

Below are some unsuccessful maps:

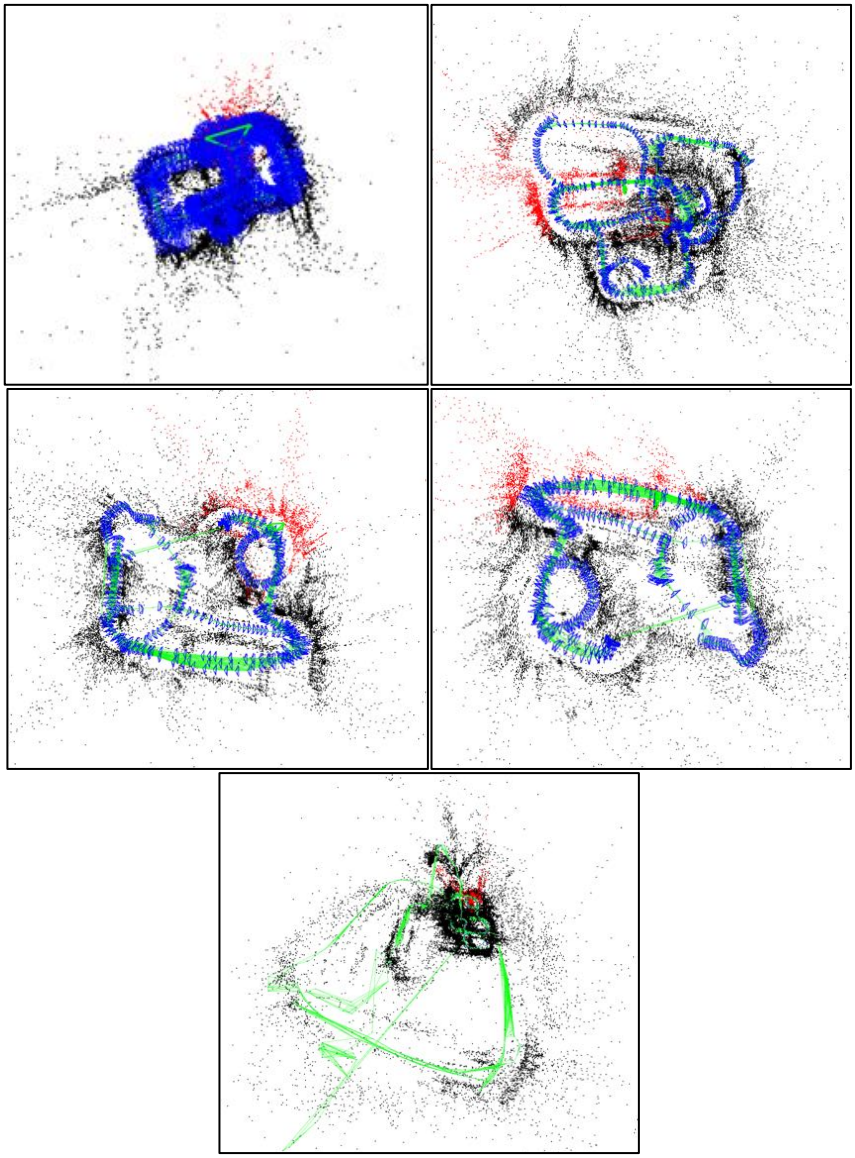


Figure 30 - Unsuccessful maps

In comparison to the successful maps, it is clear to see that the maps above were not successful. Loop closure was not achieved, and paths are not complete. The map on the top right failed during loop closure and collapsed in on itself. On the contrary, the last map drastically expanded during loop closure. Most of these maps were results of trying to create more complex structures.

A more in-depth evaluation of the inner workings of ORBSLAM can be found in the following papers [19], [20].

7 Navigation

ORBSLAM produces a 3D map. However, for navigation and path planning on a wheeled robot, only 2D movement is relevant. Due to the limitations of the monocular system, 2D mapping presents many issues.

An early navigation approach before laser sensors became standard was to use visual navigation, based on defined waypoints, using the robot's position rather than a 2D map. This works by taking images at different locations to create waypoints. To travel from one waypoint to another, the robot moves in the direction that reduces the difference between its current perceived image and the desired target image at the goal. While ORBSLAM provides the actual position of the robot, not just comparing two images for differences, a similar approach was used to overcome several issues, such as the lack of path planning approaches using only visual information. Hence, a topological map was constructed and used to navigate the environment.

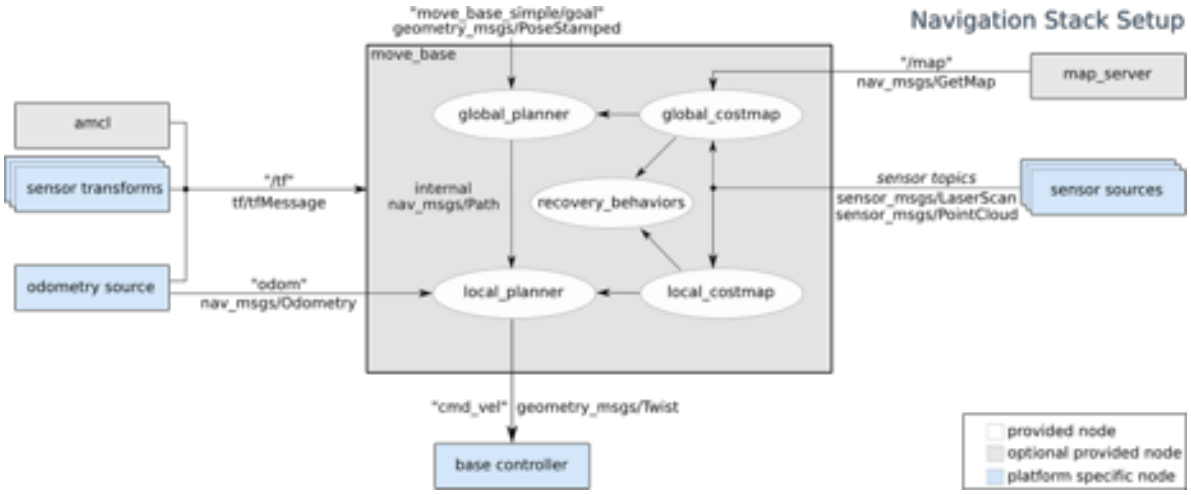


Figure 31 - Navigation stack setup

The figure above, in contrast to visual navigation, shows the typical setup for the ROS navigation stack which is widely used in the robotics community and mentioned here for the sake of completeness and to highlight the similarities and differences to the approach developed for this thesis. The sensor sources correspond to any sensory data gathered from the environment, for example laser values and camera inputs. Due to the limitations of Cozmo, the only sensor topic used in this project is the camera information. The map server is not a necessary input source for the ROS navigation stack, however it can be accommodated. In this case, due to limited sensor sources, ORBSLAM provides this information for the navigation. The general ROS navigation stack uses AMCL (adaptive monte-carlo localisation), but as described in the following section, a slightly different approach to MCL has been executed. The camera and MCL tf trees are calculated and used as an input,

along with the odometry. All of this is input into the global planner developed for Cozmo. With multiple input sources, cost maps can be used, however this is not possible in a monocular system. The local planner in the ROS navigation stack is used for local obstacle avoidance, which, once again is not possible with a monocular system. Therefore, all navigation is executed in the global planner. However, recovery behaviours are possible; due to the use of a topological map, edges can be adapted which is described as part of the behaviours in Section 8.1.

7.1 Implementation

The navigation framework for Cozmo, like for any other robot, consists of two main parts, the localisation and the navigation. As mentioned above the localisation uses a Monte-Carlo approach to fuse the information coming from the odometry and ORBSLAM. The navigation uses a topological approach where global planning happens on a graph structure of waypoints and local navigation assumes a straight line between these waypoints. The separate components are described in the following, attending to *Objective 3.*

7.1.1 Monte-Carlo Localisation

Although ORBSLAM provides a good foundation for navigation, the robot can lose localisation when turning corners or driving too close to a "building" and therefore navigation is adversely affected. To counteract this, a particle filter has been implemented. A particle filter is used to track the current position of the robot and is updated based on its odometry and camera pose, as calculated by the robots' wheel encoders and ORBSLAM respectively. The purpose of a particle filter is to 'approximate the posterior of a sample states or particles' [52]. Particle filters are successful in low dimensional spaces, and therefore provide assistance for the localisation of robots.

First, the particles are initialised with a position in the map once the first message from ORBSLAM is published, see initialise_particle_filter in *Algorithm 1*. Once the message is received, a Gaussian (or Normal) distribution in the x any y directions is created with sigma=0.1m, and particles are randomly sampled. A 1D Gaussian distribution is also created for the rotation around the z-axis (theta) with sigma=0.2rad and new values are sampled.

In order to be able to predict where the robot will move in-between receiving the message from the odometry and ORBSLAM, it assumes a constant velocity. Whenever a new odometry message is received, the constant velocity model is updated based on the current and previous position according to the odometry. See update_motion_model in *Algorithm 1*.

The main prediction loop (see predict_motion in *Algorithm 1)* runs at 30hz (3 times faster than the odometry and ORBSLAM) and uses the constant velocity model to predict the movement of the particles between messages. This increased publishing rate allows for faster position updates and supports a more reactive navigation.

Whenever a new ORBSLAM pose is received, a list of weights for the particles is calculated based on the distance of the particle to the observed ORBSLAM pose, see observe_new_pose in *Algorithm 1*. A Gaussian distribution is used to create the values for the weights, biasing them to be clustered closer to the observed pose. Once the weights are updated, new particles are drawn, with replacement based on those weights, see resample in *Algorithm 1*. To prevent particle "starvation" where the particle cloud condenses to a few particles and, therefore, becomes worse at generalising and dealing with abnormal situations, a so-called starvation factor is used, that defines the percentage of particles drawn, based on the calculated weights. The remainder of the particles are drawn randomly, similar to during initialisation, but based on the current position rather than the ORBSLAM position. The addition of the MCL should enable the exploration of waypoints that have not been mapped. This will be investigated in the following sections.

Algorithm: Monte-Carlo Localisation

Procedure: initialise_particle_filter(pose, num_particles)
for_each num_particles: # Create lists of particles for each value using a normal distribution
    x_par = [x_par, N(pose.x, 0.1)]
    y_par = [y_par, N(pose.y, 0.1)]
    t_par = [t_par, N(pose.theta, 0.1)]

Procedure: update_motion_model(Δx, Δy, Δtheta, Δt)
normalising_factor = 1/dt # normalise Δ to m/s and rad/s
velocity_x = Δx * normalising_factor
velocity_y = Δy * normalising_factor
velocity_theta = Δtheta * normalising_factor

Procedure: predict_motion(Δt)
x_par += velocity_x * Δt
y_par += velocity_y * Δt
t_par += velocity_x * Δt

Procedure: observe_new_pose(x, y, theta)
for_each px in x_par:
    ωx = [ωx, N(px|x,0.1)]
for_each py in y_par:
    ωy = [ωy, N(py|y,0.1)]
for_each pt in t_par:
    ωt = [ωt, N(pt|theta,0.1)]
ω = sum(ωx, ωy, ωt) # Creating an array of combined weights
ω = 1/sum(ω) # Normalising weights to 1

Procedure: resample(pose, num_particles, starvation_factor)
n = num_partciles * (1 – starvation_factor)
for_each n:
    x_par = draw new particle from x_par based on ω
    y_par = draw new particle from y_par based on ω
    t_par = draw new particle from t_par based on ω
initialise_particle_filter(pose, num_particles-n)

Algorithm 1 - Monte-Carlo Localisation

7.1.2 Path Planning

Two action servers were constructed for navigation, one for the waypoint navigation and one for topological navigation. The navigation between waypoints assumes them to be connected on a straight line. The tf frame published by the MCL is used to find the angle between the forward pointing x axis of the robot frame and the target waypoint using atan2. This angle (-

theta*0.6) is used as the angular velocity to orient the robot towards its next goal. A constant linear velocity of 0.03 m/s is set, until the waypoint is reached. Using 60% of the angle as the angular velocity ensures localisation is not affected. If the robot turns a corner too quick, ORBSLAM loses detection of features and therefore localisation is lost. This percentage was found via trial and error. The actual control of the robots is handled by the Cozmo driver which is provided with a twist message containing the described angular and linear velocities.

A topological map is manually composed by placing nodes (or waypoints) at intersections. Waypoints are created by saving the co-ordinate locations, connecting them via edges, and giving them unique names. These are saved in a yaml file, for the topological navigation server. In this yaml file, each waypoint is identified by a unique name, its position, and a list of other waypoints it connects to. Below shows the constructed waypoint layout and the corresponding edges and names.

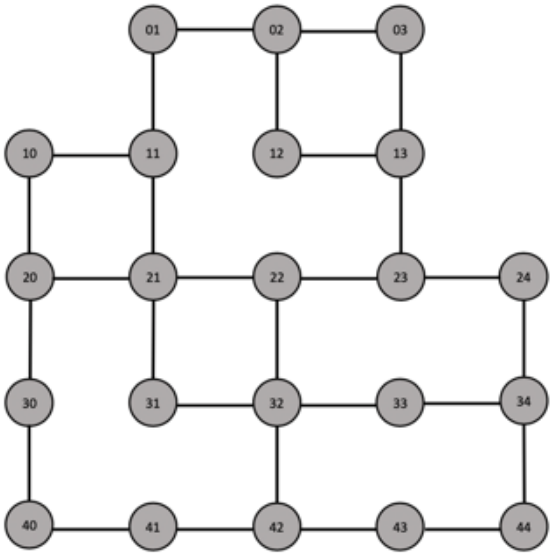


Figure 32 - Waypoint graph

Dijkstra's shortest path algorithm is implemented for path planning. This finds the shortest path between nodes in a graph and is therefore used to find the shortest path between the current location of the robot and the goal waypoint. A* search is arguably the most efficient search algorithm, however due to the distance, and therefore weight, between each node being equal, Dijkstra's shortest path works just as well but with less computation.

Dijkstra's algorithm maintains an estimate of the length of the shortest path for each edge. The algorithm processes each edge, one by one, finding new routes as it goes, and updates as necessary until the shortest route is found. For example, to traverse from waypoint 01 to waypoint 32 in figure 29, the algorithm will calculate the cost of each traversable route to the goal, then the shortest route will be the chosen result.

Once the path is calculated, the topological navigation node creates an action client for the waypoint navigation server. The client is called for every step of the path and waits until

completion, i.e. until the robot arrives at the given waypoint, this process ends; when the final goal is reached.

7.1.3 Performance of Navigation

Three experiments were carried out to test the performance of the navigation. The first experiment uses the Monte-Carlo Localisation, the second experiment uses ORBSLAM on its own for navigation, and the third experiment measures the navigation of unknown graph edges using Monte-Carlo Localisation. Each experiment is carried out for three different map routes.

The figure below shows the mapped area and the direction in which each segment was mapped. The dotted lines show areas of the environment that were not mapped. This was done previously.

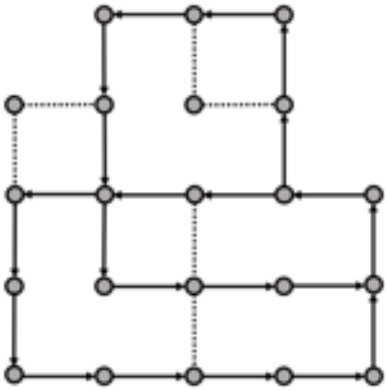


Figure 33 - Map for experiments

The figure below shows the three loops in which the navigation experiments will be carried out. There are three test loops; the green route, the blue route and the red route. For each route, the robot is placed at a waypoint in the loop and is expected to navigate through the environment until it reaches its start position.

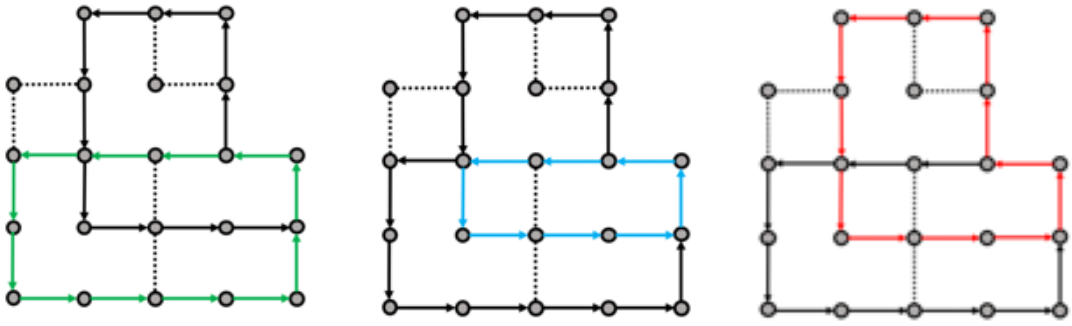


Figure 34 - Routes for navigation experiments

Each experimental condition was executed 10 times, to gather an idea of how well the system performs. The results of which are shown below. During the experiment, visual data was recorded.

7.2 Results

The technical implementation of the navigation and localisation yields a combined transformation tree (see below) that allows the robot to calculate the positions of objects perceived with its camera in the map coordinate frame, to find their absolute position. The mcl_pose frame on the other hand, is used to calculate the position of the next waypoint in relation to the current position of the robot. The same can be achieved via the camera_pose frame, however, this frame is not updated should the robot lose visual localisation. Hence, this framework and the resulting transformations allow the robot to navigate unmapped areas of the environment with reasonable success.

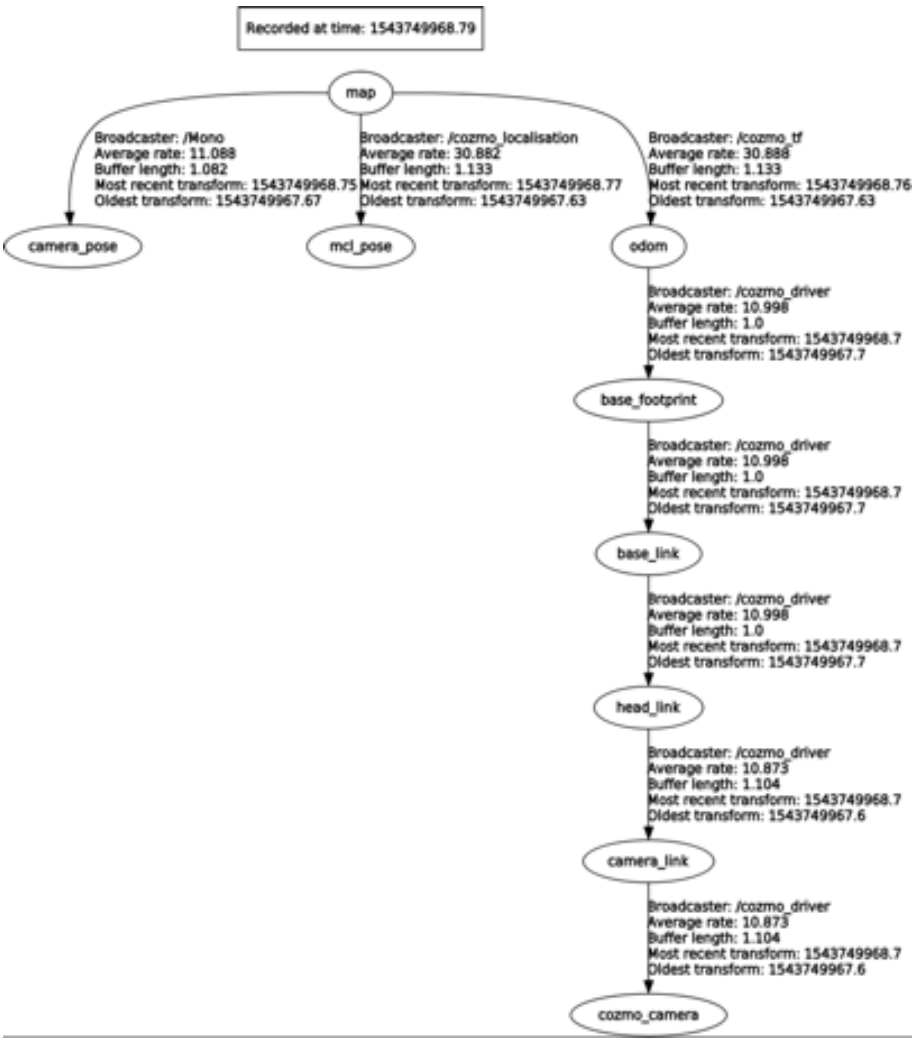


Figure 35 - Navigation tf tree

7.2.1 MCL navigation

Table 2 shows the results of navigation using MCL and ORBSLAM for the red, green and blue route shown in figure 33. The experiment was conducted 10 times for each route, with the navigation only failing once in the red route, and twice in the blue route. Each failure was at a waypoint that involved a rotation of around 90 degrees.

Table 2 - Navigation experiment 1

	Red	Green	Blue
Success Rate /10	9	10	8
Fail point	WP23	-	WP34, WP21

7.2.2 ORBSLAM navigation without MCL

Table 3 shows the results of navigation using ORBSLAM without MCL, for the red, green and blue route shown in figure 33. For this configuration, the navigation did not fail for the 10 attempts on each route. Therefore, for the routes tested, the success rate is 100%.

Table 3 - Navigation experiment 2

	Red	Green	Blue
Success Rate /10	10	10	10
Fail point	-	-	-

7.2.3 Navigating unknown paths

Due to the success of navigation in mapped environments, it was interesting to see how the system performed on an unknown graph edge.

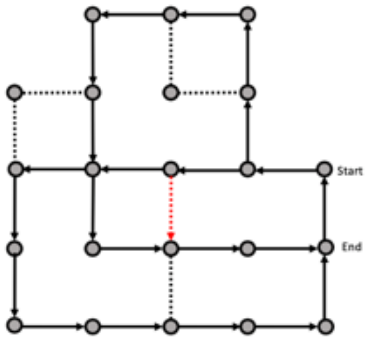


Figure 36 - Navigation experiment 3

The figure above shows the start and end position for this experiment, and the red graph edge describes the edge connection added to the topological map. The agent has no apriori information for this graph edge, other than knowing waypoint 22 has a connection to waypoint 32. Therefore, Dijkstra's algorithm will recognise this edge as a traversable path, and the shortest route from the *Start* position to the *End* position will require traversal of the new edge. This was attempted 10 times and the table below shows the results of this. Table 4 shows that the goal was reached 50% of the time.

Table 4 - Navigation experiment 3

Attempt	Success?	Fail point
1	Y	-
2	Y	-
3	N	32
4	N	22
5	Y	-
6	Y	-
7	N	33
8	Y	-
9	N	34
10	N	32
Percentage	50%	-

The results shown in this section will be further discussed in Chapter 9.

8 Behaviour Generation

This section outlines the goals generated and executed by the agent. The description of the goal, the social behaviours and the movement of the agent are described, along with the results. This section fulfils *Objective 5* and *Objective 6* concurrently.

Three different scenarios have been generated to demonstrate goal generation and behaviour. Each scenario was specifically chosen to be relevant to the proposed problem of deploying robots into hazardous environments. A summary of each is shown in the table below.

Table 5 - Summary of goal experiments

	Scenario 1	**Scenario 2**	**Scenario 3**
Goal	Re-plan routes	Search and rescue	Patrol and monitor
Task	Navigate unexpected road obstacles and re-route to reach goal	Differentiate between alive and injured people in the environment and assist where necessary	Patrol the environment and perform visual inspections of gauges
Behaviour	Adaptive	Reactive	Goal-driven
Feedback	Path planning	Verbal and motor control	Verbal, motor control and reporting

Each scenario is performed using the integrated system, of which components have been described throughout this report, as a proof of concept. The fundamental objective of each goal is to:

- Navigate
- Communicate
- Execute actions
- Monitor/report

8.1 Scenario 1 - Road Block

The first task for the agent is to modify the route based on road blocks. In lieu of obstacle detection using a laser scanner or similar, the road block is indicated with a marker, shown in the figure below. If this marker is detected close to the next waypoint in the path the robot is supposed to take, i.e. 10cm from the waypoint as calculated using the Euclidean distance between the tf information from the marker and the location of the waypoint, the edge is assumed to be non-traversable.

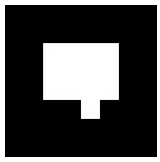


Figure 37 - Road block marker

The agent navigates the environment from the start waypoint to the goal waypoint, if a road block is seen and the edge is assumed non-traversable, the agent must plan a new route and execute it. Since monocular navigation doesn't allow dynamic obstacle avoidance and local path planning, the only way to reach the goal is to change the global path. In order to achieve this, the topological navigation was modified to detect these blockages and remove non-traversable edges. Once the edge is removed, Dijkstra is run again to find an alternative path. This is repeated until the goal is reached.

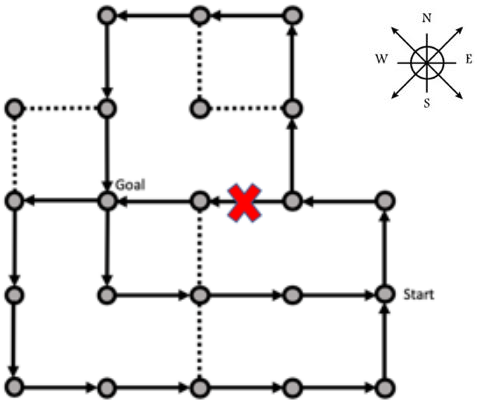


Figure 38 - Road block experiment setup

The figure above shows the start position, the goal, and the red cross shows the road block. As you can see in the waypoint graph, there are two executable routes to the goal. The first route being: N-E-E-E with a cost of 4, and the second being N-E-N-N-E-E-S-S with a cost of 8. Due to the implementation of Dijkstra, the chosen route would be the one with the lowest cost. However, with the road block being placed at the red X, this blocks the chosen route. Therefore, when the agent reaches this point, it should reroute and execute the second route described. This experimental task was executed 10 times and the results can be seen in the section below.

Algorithm: Adaptive Path Planning
Input: goal

Procedure: dijkstra(goal, origin)

```
    set costs of all nodes to infinity
    set costs of origin to 0
    mark all nodes as open
    while N != goal:
        find open node N with lowest estimated costs
        close(N)
        for_each M in edges(N):
            if open(M):
                new_cost = cost(N) + 1 # All movements have the same costs
                if new_costs < cost(M): cost(M) = new_cost
                link(N, M)
    path = follow links from goal to target

origin = estimate current location based on Euclidean distance to known waypoints
path = dijkstra(goal, origin)

for_each waypoint in path:
    if close(obstacle_marker, waypoint):
        remove edge between current and next waypoint
        path = dijkstra(goal, current waypoint)
    call and wait for waypoint_navigation(waypoint)
```

Algorithm 2 - Adaptive path planning

8.1.1 Results

During the experiment, visual data was recorded. Due to the nature of the data, a percentage success rate can be calculated. This is shown in the table below. The key components to be measured where whether the robot detected the road block, whether a new route was calculated, whether the final goal was accomplished, and whether the robot verbally announced its discoveries.

Table 6 - Road block experiment results

Attempt	Road block detected? Y/N	Re-route calculated? Y/N	Goal reached? Y/N	Speech behaviour performed? Y/N
1	Y	Y	Y	Y
2	Y	Y	Y	Y
3	Y	Y	Y	Y

4	Y	Y	Y	Y
5	Y	Y	Y	Y
6	Y	Y	Y	Y
7	N	N	Y	N
8	Y	Y	N	Y
9	Y	Y	Y	Y
10	Y	Y	Y	Y
Percentage	90%	90%	90%	90%

As seen in the table above, the experiment produced a success rate percentage of 90% for each measurement.

8.2 Scenario 2 - Search and Rescue

The second task for the experiment is to search and rescue injured 'personnel' from the environment. If a person is spotted, the agent should check if they are injured or not. If not, the agent should politely greet the 'person' and continue. If the agent discovers an injured person, the agent should 'call' the emergency services, and stay at the scene until the emergency services arrive. Once the injured person goes, (i.e. the injured person is removed from the scene) the agent should continue on with navigation. The markers shown in the figure below describe the marker indicating an 'injured person' and an 'alive person'.

Figure 39 – Markers. Left: injured person, Right: alive person

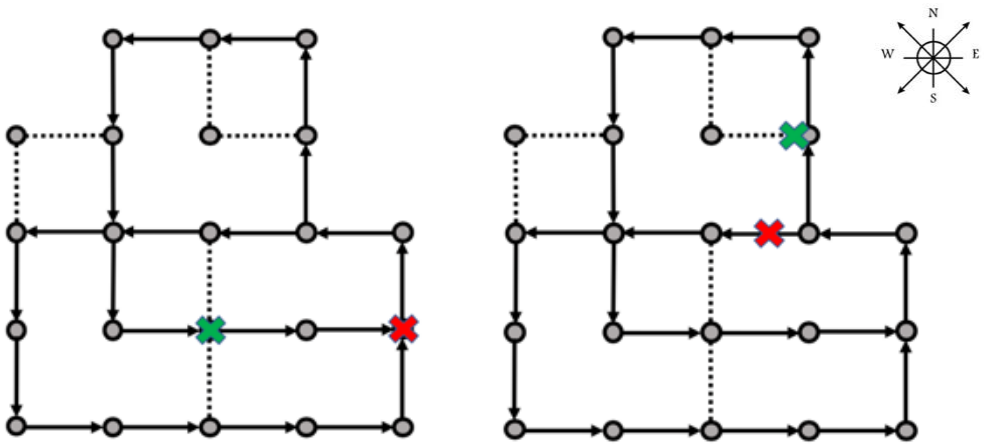


Figure 40 - Search and rescue experiment setup

The figure above illustrates the positioning of the markers in the environment, for two different configurations. The green X marks an 'alive' person, where the red X marks an 'injured person. Each configuration was tested five times; The configuration on the left was used in attempt 1-5, and the configuration on the right used for attempt 6-10 in the results section below.

Algorithm: Search and Rescue
Input: goal

Procedure: check_for_people()

```
while True:
    markers = get current AR markers
    for_each marker in markers:
        if id(marker) == healthy:
            greet human
        if id(marker) == injured:
            stop navigation
            call emergency services
            while visible(marker):
                wait
            resume navigation
            break

call Adaptive Path Planning(goal)
check_for_people()
```

Algorithm 3 - Search and rescue

8.2.1 Results

Figure 41 – Search and rescue ROS graph

The figure above shows a graph of the connected topics for this experiment. The key components measured where whether the robot detected the alive person and whether it produced the appropriate response, whether the robot detected the injured person and produced the appropriate response, and whether the navigation goal was resumed and completed.

Table 7 - Search and rescue results

Attempt	Alive		Injured		Task completed?
	Seen?	Behaviour?	Seen?	Behaviour?	
1	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y
3	Y	Y	Y	Y	Y
4	Y	Y	Y	Y	Y
5	Y	Y	Y	Y	Y
Percentage	100%	100%	100%	100%	100%
6	N	N	Y	Y	Y
7	N	N	N	N	Y
8	Y	Y	Y	Y	Y
9	N	N	Y	Y	Y
10	Y	Y	Y	Y	Y
Percentage	40%	40%	80%	80%	100%

As seen above, the initial configuration achieved a success rate of 100%. Therefore, it was decided to rearrange the positioning of the 'people' to evaluate if the positioning effects the results. The second configuration provided significantly worse results, with the success rate of seeing the alive person being reduced to 40% and the success rate of seeing the injured person reducing to 80%.

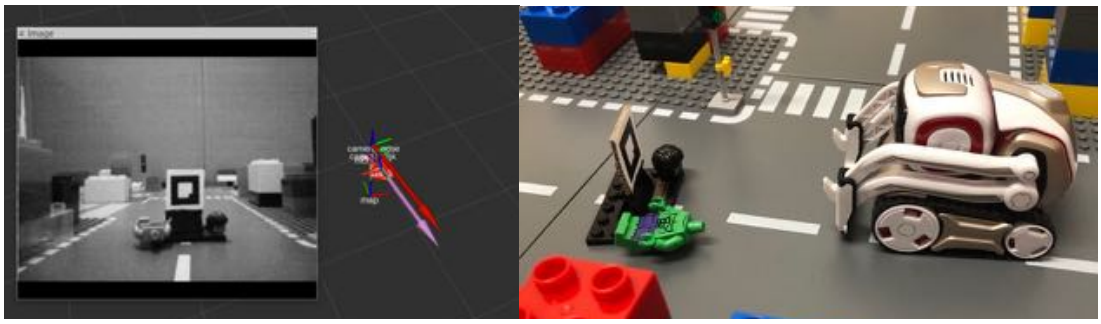


Figure 42 - Detection of injured person

The figure above shows the ‘injured person’ from the perspective of the robot, and the robot with the ‘person’ in the environment. The figure below shows the same but with the ‘alive person’.

Figure 43 - Detection of alive person

8.3 Scenario 3 - Patrolling and Monitoring

The third task tested was for the agent to patrol the environment, monitoring and checking ‘gauge values’ when passing them. If a ‘gauge’ is discovered, the agent will verbally announce the state of the situation. If any abnormal readings are found, this should be reported when the agent reaches its navigation goal.

The figure below displays the markers used in this experiment. The marker on the left indicates that a gauge is present, where the marker on the right indicates an abnormal gauge reading.

Figure 44 - Markers. Left: gauge, Right: abnormal reading

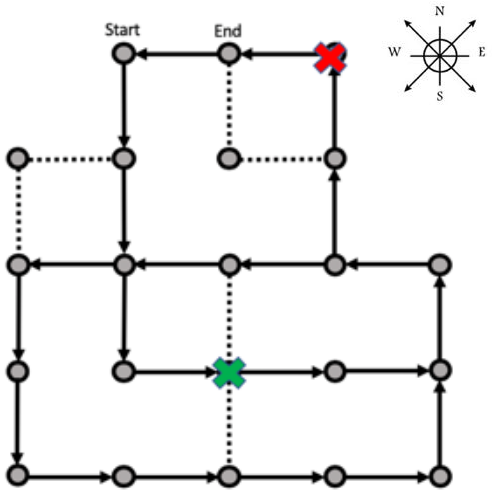


Figure 45 – Patrolling experiment setup

The X's in the figure above represent the positions of the 'gauge's' to be monitored. The green X representing a 'normal' reading and the red X representing an 'abnormal' reading. The experiment consisted of the robot patrolling the area, starting from the position indicated in the figure above, and ending at the indicated position. When the robot reaches the end position, it should report its findings. This was conducted 10 times and the recorded results are shown below.

Algorithm: Gauge Check
Input: goal

Procedure: check_gauge()

```
    while task active:
        markers = get current AR markers
        for_each marker in markers:
            if id(marker) == gauge:
                stop navigation
                fault = False
                for_each marker in markers:
                    if id(marker) == abnormal_value:
                        fault = True
                        faults += 1
                if fault:
                    give warning
        resume navigation
    return faults

call Adaptive Path Planning(goal)
faults = check_gauge()
report faults
```

Algorithm 4 - Gauge check

8.3.1 Results

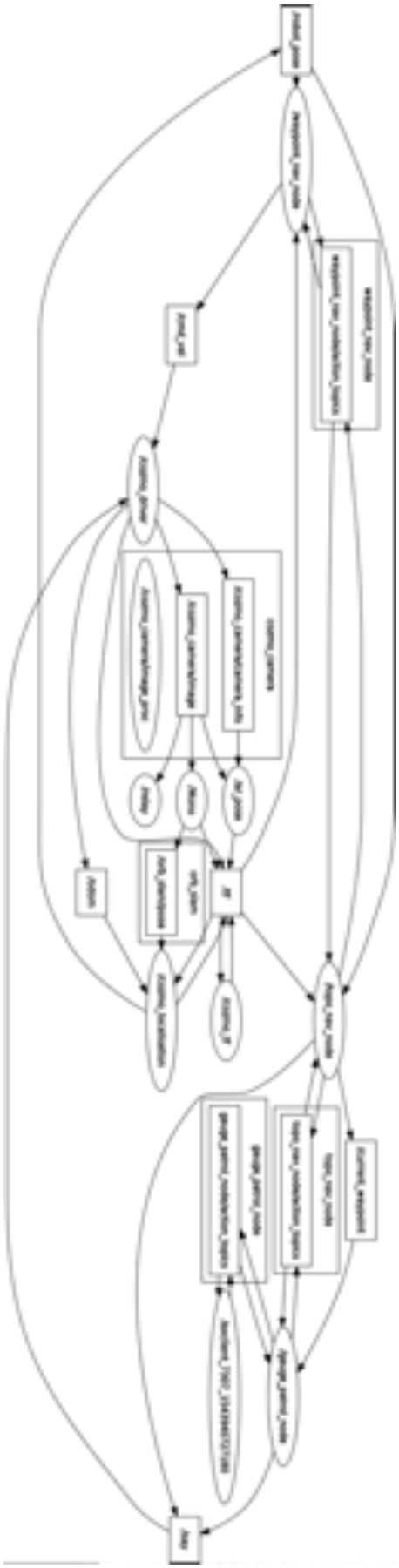


Figure 46 - Patrol and monitor ROS graph

The figure above shows a graph of the connected topics for this experiment. The key components measured where whether the robot detected the gauges and whether it produced the appropriate behavioural responses. Any false marker detections were also recorded, along with whether the robot reported its finding once it reached its navigation goal.

Table 8 - Patrol and Monitor results

Attempt	See 1st gauge	Correct behaviour?	See 2nd gauge	Correct behaviour?	Report at goal	False detection?
1	Y	Y	Y	Y	Y	Y – WP13
2	Y	Y	Y	N	N	N
3	Y	Y	Y	Y	Y	N
4	N	N	Y	Y	Y	N
5	Y	Y	N	N	N	Y – WP13
6	Y	Y	N	N	Y	N
7	Y	Y	Y	Y	Y	N
8	Y	Y	Y	Y	Y	N
9	Y	Y	Y	Y	Y	N
10	Y	Y	Y	Y	Y	N
Percentage	90%	90%	80%	70%	80%	20%

The table above shows the results from running the experiment 10 times. The 'working gauge' was detected with a success rate of 90%, where the 'abnormal gauge' (2nd gauge) was detected with an error rate of 80%. Only two false detections were made, and the results were reported at the goal 80% of the time.

9 Discussion

Based on the research undertaken and the results of implementation, the questions raised in the introduction can now be addressed. The system can benefit humans in the event of a disaster by patrolling and monitoring, detecting injured people and detecting abnormal observations. The system successfully allows for detecting the nature and location of hazards in the environment and reporting them when necessary. The use of the navigation framework implemented allows to determine the safety of the route to be taken, and to avoid obstacles if necessary.

9.1 Image filtering

As shown in Chapter 4, a Gaussian filter and a Bilateral filter were applied to the raw image data from Cozmo. Both the Gaussian filter and the Bilateral filter reduced the noise in the image, however when both filters were applied together, the image was too blurred for ORB feature matching. Therefore, the filter used was the bilateral filter, with the parameters being (5, 75, 75). This was preferential over the Gaussian filter, because the bilateral filter is essentially an extension of the Gaussian filter. It still provides the blurring that the Gaussian filter provides, but with clearer, sharper edges which is advantageous for SLAM.

9.2 SLAM

As described in Chapter 6, mapping the environment presented an extensive and time-consuming process. These were mainly due to scaling, errors with loop closure, and losing localisation. For example, in order to create the map for the navigation experiments, it took 8 attempts to produce the map shown below in figure 47. Successful maps, such as the one below, gave a clear depiction of the environment that was mapped.

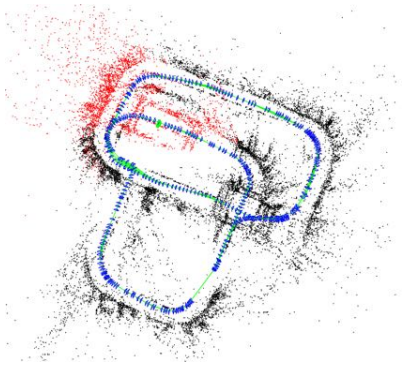


Figure 47 - Successful map

The successful maps tended to be the simpler shapes without too complicated routes. ORB feature detection often failed when traversing around 90-degree angled corners, therefore localisation was lost, and the map subsequently failed. Further to this, the scaling factor changes depending on the robots' location in the map, which has an adverse effect of loop closure. Due to time restrictions, it was not possible to attempt mapping on the oil rig, nevertheless, the system should be accomplishable with only minor alterations needed to parameters such as the scaling factor.

The main limitation of ORBSLAM is that it doesn't take odometry into account, so mapping isn't scaled correctly, meaning that multi-directional mapping doesn't associate the paths with each other.

9.3 Navigation

The experimental results show that the success rate of navigation was higher without the use of the MCL. This is due to the MCL co-ordinate frame being based on odometry, and therefore the actual movement that the robot undertook, rather than the ORBSLAM co-ordinate frame, which is based purely on visual information. Therefore, the ORBSLAM co-ordinate frame isn't influenced by any difference in scale between the map and the real world. However, during the experiments, ORBSLAM lost localisation in certain situations, but it was able to recover quickly enough for the navigation not to fail. In practice however, this cannot be guaranteed to be the case, given possible changing environments and lighting conditions etc.

The benefit of MCL is that it also enables navigation in unknown environments, as can be seen in table 4. Although the success rate was only 50%, the current ORBSLAM system on its own does not allow for this at all. This could be improved by using a confidence value for visual localisation. This is not provided by ORBSLAM at the moment, however it could be implemented in the MCL particle filter, influencing the weights for re-sampling the particles.

The main limitation with regards to topological navigation is that the waypoints were required to be connected on a straight line, therefore it is impossible to do dynamic obstacle avoidance. This stems from the limitation of monocular SLAM systems and visual navigation. Another possible reason for failures in navigating being slippage, due to Cozmo being a tracked vehicle. For example, if Cozmo gets stuck close to a building, the wheels keep moving, while Cozmo doesn't, therefore the odometric error increases.

9.4 Scenario 1

For the adaptive navigation, table 6 shows a 90% error rate, where in one occasion the AR marker was not detected, and therefore no re-planning was carried out. However, Cozmo still reached its goal by simply pushing the marker, which almost completely blocked its vision. Thanks to MCL and the robustness of ORBLAM, it was still able to navigate towards its goal. The table also shows that the navigation failed to reach the goal on one occasion, despite it

successfully detecting the marker and re-planning. This is due to a navigation failure, as described above. In general, these results show that this kind of obstacle avoidance via changing the global plan to re-route the robot is a successful strategy to avoid impassable areas and road blocks.

9.5 Scenario 2

As mentioned in the results in section 8.2.1, it can be seen that if the markers are placed in locations that they can be easily detected, the implemented approach has a perfect success rate. This shows that it works well in principle. However, if the markers are placed in locations that might only be seen while the robot turns, i.e. subject to motion blur or only visible for short times, the approach fails due to not detecting the marker.

9.6 Scenario 3

Similar to scenario 2, if the markers were detected, the behaviour worked as expected except for two occasions where the status of the gauge's was not reported. This was due to the robot not reaching its goal location, at which it was supposed to give the report. This was due to a navigation error, as described above. On all other occasions where it failed, these were due to either false negatives or false positives in the marker detection. On one occasion, a road block detected where there wasn't one, and therefore tried to re-route, and didn't see the markers.

10 Conclusion

This project intended to design and execute a cognitive architecture on a small-scale robotic platform for a proof of concept, assisting in the employment of robotics for hazardous environments. An architecture was designed and implemented for application on Cozmo using ROS. Appropriate methods were adopted; the state-of-the-art monocular SLAM, ORBSLAM was integrated, topological navigation with Monte-Carlo localisation was implemented, and AR marker object detection was combined to facilitate goal generation and execution for three different environmental scenarios.

The results of the experiments show that the system successfully meets the objectives outlined at the beginning of this report. Moreover, due to the fact that all implementation was carried out using ROS, it can be easily transferred onto other robots and extended to more complex scenario's and applications. The fact that the system only relies on a single camera as a sensor further supports the fact that it can be deployed on almost any robot available nowadays.

10.1 Future Work

As this project has focused on the visual perception property of a cognitive architecture, future work would include the design and integration of reasoning and learning. For example, if the agent is faced with multiple issues at once, it would be useful to have some form of decision making for prioritisation of tasks. Furthermore, multiple agents could be deployed into hazardous environments, with the aim to 'divide and conquer' using swarming techniques.

Although the AR markers work as a proof of concept for object recognition, it would be beneficial to implement more specific object recognition algorithms, such as 'Open Pose Estimation' for person detection.

As discussed previously, the main limitation to monocular SLAM is the lack of sensory information for obstacle avoidance, therefore it would be beneficial to investigate possible approached to combat this. Another possible development for ORBSLAM would be to incorporate the odometry of the robot to counter the scaling problem when using monocular vision.

With regards to the topological navigation system developed in this project, the waypoint graph constructed is unidirectional. With the simulated environment providing 'lanes' on the 'road', it would be interesting to investigate the possibilities of a multi-directional system to incorporate the lanes. In order for this work, above suggestion of incorporating the robot's odometry into ORBSLAM would be crucial. Otherwise, the lanes might not be correctly associate with each other in the map.

11 Other Work

11.1 RoboCation

A cognitive architecture is an interesting concept to be considered in teaching. Each of these topics can be implemented in different ways as modules for a bigger system. This can be useful for teaching purposes, as individual elements can be extracted, modified and investigated independently of the rest of the system. For example, learning could be investigated; the learning algorithm implemented could be associative, where the decision-making process is influenced by reward and punishment, or it could be procedural, where the learning skills progress gradually through repetition until the skill becomes automatic.

11.1.1 Teaching Framework

The framework proposed is based on the 'Building blocks for learning – learning objects' idea designed by the *MASIE Center* [53]. Figure 48 as a whole, can be considered as a **Course**. Each course is made up of **Lessons**. Each lesson is constructed from **Learning Objects**. Learning objects are constructed of **Information Objects** that meet a single objective. Information objects are made up of things such as Concepts, Facts, Principles, Processes, Procedures, Overview, Summary. These are constructed from **Raw Content Items** such as text, media, images etc.

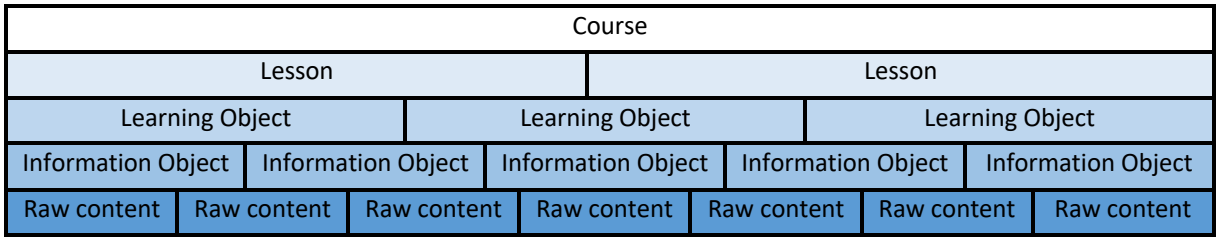


Figure 48 - Learning framework

Raw content items are the most re-usable but have the least context. The can be used for any part of the framework. Courses and lessons are the least re-usable but contain the most context. They are application specific and based on a theme/topic. The idea is that you do not need to learn all of each colour to progress to the next level. It is dependent on the topic.

A learning object can also be defined as a tutorial. In each learning object, a tutorial for the student and a tutorial for the teacher can be created. An example tutorial could contain the following information objects: A learning goal, a brief background, setup instructions, task outline, task instructions and a summary. These will be subject specific.

Cozmo has an already existing application, which supplies an interactive interface with different levels of complexity. This ranges from playing games and interacting, to programming simple code in 'scratch'. Therefore, each level of complexity can be used for different skill and experience levels. Figure 49 shows each level to be considered for teaching. This adds a 'third dimension' to the learning structure in figure 48.

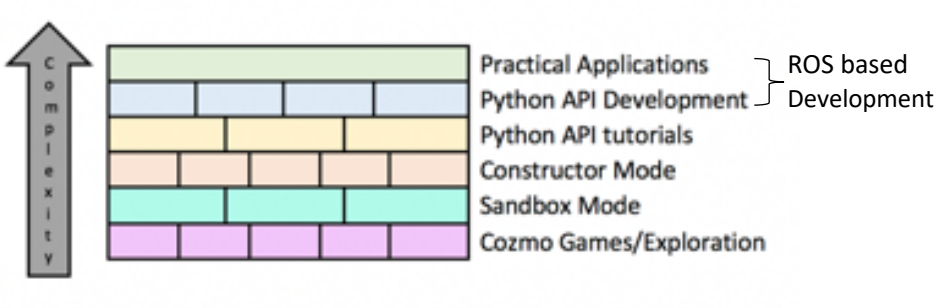


Figure 49 – Teaching levels

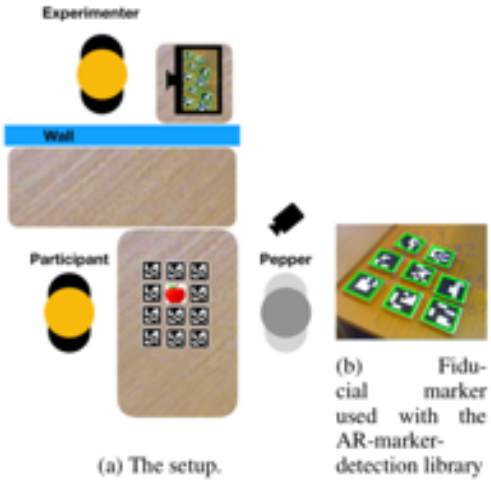
11.1.2 Branding

Figure 50 shows the logo designed for *RoboCation*. This was designed in such a way to appeal to all ages, whilst incorporating the learning aspect. This logo will represent *RoboCation* and will be featured on all marketing, documentation and learning material.

Figure 50 - RoboCation Logo

11.2 Human Robot Interaction Experiment

Two MSc students under the supervision of Dr Katrin Lohan, required assistance in executing an HRI experiment. Each student had a different research issue; one students' focus was to investigate cognitive load during an intellectually challenging task, whilst the other student had the focus of investigating the effects of trust and anthropomorphism in human-robot interaction. These were combined into one HRI experiment. The setup can be seen in figure 51.



[54]

Figure 51 - Experimental setup

The experiment was designed around playing a game called 'matching the pairs', where the participant would have to match pairs from 12 blocks, turning one over at a time and asking for help from the robot if required. Participants wore Tobii 2 glasses while playing the game, to measure their cognitive load.

Two robots were used; Husky, a machine-like robot, and Pepper, a humanoid robot. This provided the variables for measuring the effects of anthropomorphism and trust. The participant was able to verbally ask the robot for help during the game, which would instigate the robot to suggest the matching pair. For each robot, there were two different error rates, 3% and 50%. These error rates were used to measure how the participant would react if the robot gave them an incorrect suggestion.

Over a period of 3 weeks, 40 participants took part in the experiment, 20 per robot, split into 10 for a 3% error rate, and 10 for a 50% error rate. Each participant was asked to fill in a pre-test questionnaire, play the game, then fill in a post-test questionnaire. The questions were designed to analyse how the participant felt about interacting with the robot and if they would trust the answers that the robot provided.

By helping to organise and oversee the process, I gained knowledge and hands-on experience in running an HRI experiment. I learned the significance of controlling the environment, to ensure the reliability of results, and the importance of preparation before starting an experiment like this.

Following on from this, I collaborated with colleagues to write a paper centred on the trust and cognitive load during human-robot interaction. This has been submitted as a full paper submission to Alt-HRI 2019, and a copy of the paper can be found in Appendix B.

12 References

[1] H. Hastie *et al.*, "The ORCA Hub: Explainable Offshore Robotics through Intelligent Interfaces," pp. 1–2, 2018.

[2] "taurob | Robots for dangerous missions." [Online]. Available: http://taurob.com/. [Accessed: 04-Dec-2018].

[3] P. Langley, J. E. Laird, and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cogn. Syst. Res.*, vol. 10, pp. 141–160, 2009.

[4] I. Kotseruba and J. K. Tsotsos, *40 Years of Cognitive Architectures: Core Cognitive Abilities and Practical Applications*. Springer Netherlands, 2018.

[5] "ROS.org | About ROS." [Online]. Available: http://www.ros.org/about-ros/. [Accessed: 18-Nov-2018].

[6] "ROS-Industrial." [Online]. Available: https://rosindustrial.org/. [Accessed: 30-Nov-2018].

[7] "actionlib - ROS Wiki." [Online]. Available: http://wiki.ros.org/actionlib. [Accessed: 30-Nov-2018].

[8] "Anki Homepage | Explore Our Intelligent Robot Products | Anki UK." [Online]. Available: https://www.anki.com/en-gb. [Accessed: 02-Dec-2018].

[9] "iCub.org - an open source cognitive humanoid robotic platform." [Online]. Available: http://www.icub.org/. [Accessed: 18-Nov-2018].

[10] "Anki Developer Forums." [Online]. Available: https://forums.anki.com/. [Accessed: 30-Nov-2018].

[11] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," *Integr. VLSI J.*, vol. 59, no. May, pp. 148–156, 2017.

[12] C. Cigla, R. Brockers, and L. Matthies, "Image-based Visual Perception and Representation for Collision Avoidance," pp. 421–429, 2017.

[13] and C.-C. W. Chien-Ming Lin, Chi-Yi Tsai, Yu-Cheng Lai, Shin-An Li, "Visual Object Recognition and Pose Estimation Based on a Deep Semantic Segmentation Network," vol. 1748, no. c, pp. 1–11, 2018.

[14] "OpenSLAM.org." [Online]. Available: https://openslam-org.github.io/gmapping.html. [Accessed: 29-Nov-2018].

[15] G. Grisetti and C. Stachniss, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling."

[16] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, 2007.

[17] Y. Kameda, “Parallel Tracking and Mapping for Small AR Workspaces (PTAM) Augmented Reality,” *J. Inst. Image Inf. Telev. Eng.*, vol. 66, no. 1, pp. 45–51, 2012.

[18] T. B. Ludermir, C. Zanchettin, and A. Carolina Lorena, *Advances in intelligent systems*, vol. 127. 2014.

[19] J. M. M. Montiel, “ORB-SLAM : A Versatile and Accurate Monocular,” vol. 31, no. 5, pp. 1147–1163, 2015.

[20] R. Mur-Artal and J. D. Tardos, “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras,” *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, 2017.

[21] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” pp. 1–28, 2004.

[22] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF : Speeded Up Robust Features.”

[23] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 2564–2571, 2011.

[24] E. Rosten and T. Drummond, “10.1.1.64.8513,” pp. 1–14, 2010.

[25] E. Rosten, R. Porter, and T. Drummond, “Faster and better: A machine learning approach to corner detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 105–119, 2010.

[26] and P. F. Michael Calonder, Vincent Lepetit, Christoph Strecha, “BRIEF: Binary Robust Independent Elementary Features⋆,” vol. 2011, no. 10/12, 2011.

[27] W. Churchill and P. Newman, “Experience-based navigation for long-term localisation,” *Int. J. Rob. Res.*, vol. 32, no. 14, pp. 1645–1661, 2013.

[28] R. World, “The Dynamic Window Approach,” no. March, pp. 23–33, 1997.

[29] D. Filliat and J.-A. Meyer, “Map-based navigation in mobile robots: II. A review of map-learning and path-planning strategies,” *Cogn. Syst. Res.*, vol. 4, no. 4, pp. 243–282, 2003.

[30] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics (intelligent robotics and autonomous agents series),” *Intell. Robot. Auton. agents, MIT …*, vol. 45, no. 3, p. 52, 2005.

[31] “navigation/Tutorials/RobotSetup - ROS Wiki.” [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Accessed: 07-Dec-2018].

[32] A. Intelligence, “Learning metric-topological maps for indoor mobile robot navigation ’,” *Artif. Intell.*, vol. 99, pp. 21–71, 1998.

[33] J. P. Fentanes, B. Lacerda, T. Krajnik, N. Hawes, and M. Hanheide, “Now or later? Predicting and maximising success of navigation actions from long-term experience,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2015–June, no. June, pp. 1112–1117, 2015.

[34] N. Winters and J. Santos-Victor, “Omnidirectional visual navigation,” in *Proc. of IEEE Int’l Symposium on Intelligent Robotic Systems (SIRS)*, pp. 109–118.

[35] J. Gaspar, N. Winters, and J. Santos-Victor, “Vision-based navigation and environmental representations with an omnidirectional camera,” *IEEE Trans. Robot. Autom.*, vol. 16,

no. 6, pp. 890–898, 2000.

[36] J. Kosecka, Liang Zhou, P. Barber, and Z. Duric, “Qualitative image based localization in indoors environments,” *2003 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition, 2003. Proceedings.*, vol. 2, p. II-3-II-8, 2003.

[37] M. Tomono, “3-D object map building using dense object models with SIFT-based recognition features,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 1885–1890, 2006.

[38] D. Schleicher, L. M. Bergasa, R. Barea, E. López, and M. Ocaña, “Real-time simultaneous localization and mapping using a wide-angle stereo camera and adaptive patches,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2090–2095, 2006.

[39] S. Thrun *et al.*, “MINERVA : A Second-Generation Museum Tour-Guide Robot,” no. May, 1999.

[40] J. Shen and O. Hu, “Visual navigation of a museum guide robot,” *Proc. World Congr. Intell. Control Autom.*, vol. 2, pp. 9169–9173, 2006.

[41] J. Courbon, Y. Mezouar, N. Guenard, and P. Martinet, “Visual navigation of a quadrotor aerial vehicle,” *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, no. November, pp. 5315–5320, 2009.

[42] T. Krajník, M. Nitsche, S. Pedre, L. Přeučil, and M. E. Mejail, “A simple visual navigation system for an UAV,” *Int. Multi-Conference Syst. Signals Devices, SSD 2012 - Summ. Proc.*, 2012.

[43] “camera_calibration/Tutorials/MonocularCalibration - ROS Wiki.” [Online]. Available: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration. [Accessed: 05-Nov-2018].

[44] “Augmented reality | computer science | Britannica.com.” [Online]. Available: https://www.britannica.com/technology/augmented-reality. [Accessed: 06-Dec-2018].

[45] “ARToolKit Home Page.” [Online]. Available: http://www.hitl.washington.edu/artoolkit/. [Accessed: 06-Dec-2018].

[46] X. Zhang, S. Fronz, and N. Navab, “Visual marker detection and decoding in AR systems: A comparative study,” *Proc. - Int. Symp. Mix. Augment. Reality, ISMAR 2002*, pp. 97–106, 2002.

[47] N. Mazouni, L. Loubersac, H. Rey-Valette, T. Libourel, P. Maurel, and J.-C. Desconnets, “Vision Algorithms: Theory and Practice,” *Vie Milieu-Life Environ.*, vol. 1883, no. 4, pp. 265–274, 2000.

[48] H. Strasdat, A. J. Davison, J. M. M. Montiel, and K. Konolige, “Double Window Optimisation for Constant Time Visual SLAM.”

[49] C. Mei, G. Sibley, and P. Newman, “Closing loops without places,” *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.*, pp. 3738–3744, 2010.

[50] H. Strasdat, J. M. M. Montiel, and A. Davison, “Scale Drift-Aware Large Scale Monocular SLAM,” *Robot. Sci. Syst. VI*, 2010.

[51] A. Benetnash, “ORBSLAM2.” [Online]. Available: https://github.com/Alkaid-Benetnash/ORB_SLAM2.

[52] S. Thrun, “Particle Filters in Robotics,” no. 2, p. 40, 1982.

[53] MASIE Centre Learning Consortium, “Making Sense of Learning Specifications & Standards: A Decision Maker’s Guide to their Adoption.” pp. 42–52.

[54] A. Yaseen and K. Lohan, “Playing Pairs with Pepper,” no. Duffy 2003, 2018.

13 Appendix

Appendix A – Camera Calibration Results

	Attempt 1	Attempt 2	Attempt 3
Image Width	320	320	320
Image Height	240	240	240
Camera Matrix	$\begin{matrix} 316.626 & 0 & 183.564 \\ 0 & 317.841 & 119.385 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 316.549 & 0 & 168.138 \\ 0 & 317.836 & 138.165 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 301.429 & 0 & 163.977 \\ 0 & 302.263 & 126.806 \\ 0 & 0 & 1 \end{matrix}$
Distortion Coefficients	k_1 0.234583 k_2 −0.137046 k_3 0.000000 p_1 0.005218 p_2 0.013470	k_1 0.149818 k_2 0.084514 k_3 0.000000 p_1 0.018515 p_2 -0.009475	k_1 0.116077 k_2 0.051566 k_3 0.000000 p_1 0.008385 p_2 -0.008439
Rectification Matrix	$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$
Projection Matrix	$\begin{matrix} 337.084 & 0 & 185.459 & 0 \\ 0 & 339.001 & 120.244 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 337.714 & 0 & 165.473 & 0 \\ 0 & 340.446 & 140.693 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 318.615 & 0 & 161.672 & 0 \\ 0 & 321.026 & 128.158 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$

	Average
Image Width	320
Image Height	240
Camera Matrix	$\begin{matrix} 311.534 & 0 & 171.893 \\ 0 & 312.647 & 128.119 \\ 0 & 0 & 1 \end{matrix}$
Distortion Coefficients	k_1 0.166826 k_2 −0.000966 k_3 0.000000 p_1 0.010706 p_2 −0.001481
Rectification Matrix	$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$
Projection Matrix	$\begin{matrix} 331.137 & 0 & 170.868 & 0 \\ 0 & 333.491 & 129.698 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$

Appendix B – HRI conference submission

This is the double-blinded review copy of the submission, and therefore does not list the authors' names. This paper is currently under review, the acceptance rate being 25%.

Trust and Cognitive Load During Human-Robot Interaction

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—This paper presents an exploratory study to understand the relationship between a humans' cognitive load, trust, and anthropomorphism during Human-Robot Interaction. To understand the relationship, we created a "Matching the Pair" game designed to enable humans to play the game collaboratively with either the Husky or the Pepper robots. The goal was to understand if humans trust the robot as a teammate under situations demanding higher cognitive load to complete a task during the game playing setup. Additionally, we also exploited anthropomorphism as a factor by enabling humans to interact with either a humanoid (Pepper) or a non-humanoid (Husky) robot possessing either low or high error-rates. We found that there was an inversely proportional relationship between trust and cognitive load, suggesting that as the amount of cognitive load increased in the participants, their level of trust decreased. Additionally, we also found that participants perceived Pepper to be more trustworthy in comparison with the Husky robot after the interaction for this task.**

***Index Terms*—Trust, Cognitive Load, Anthropomorphism, Human-Robot Interaction**

I. Introduction

With the recent advancements in the field of Artificial Intelligence and Machine Learning, there is a growing interest in applying various methods from these fields to Robotics, in order to create intelligent, user-friendly robots that could seamlessly collaborate and interact with users in the future. Similarly, we also witness a transition from robots performing a tool-based role to undertaking the role of an interdependent social actor [1]. An aim of current Human-Robot Interaction (HRI) research is to create robots that can adapt to user needs [2]. It is a common expectation in the HRI field that such robotic systems will highly influence attitudinal preferences of the individuals. These attitudinal preferences include improving their likeability, enhancing trust perception, minimizing cognitive load, willingness to work together with robots and much more.

We are particularly interested in understanding the factors that could lead to maximizing users' trust perception during HRI. Most recently, a meta-analysis on the factors affecting users' trust during HRI revealed that the robot's task performance and attributes such as anthropomorphism and proximity are among significant factors that result in enhanced user trust [3]. However, it should also be noted that this meta-analysis was based on a limited number of studies and also presented differing results on the factors affecting user trust. Most recently, Bernotat et al. (2018) also investigated German and Japanese judgments of an anthropomorphic vs. an industrial robot in smart homes. They found a marginally significant difference in Japanese vs. German participants' trust ratings, but they did not find an effect of robot type on trust. That is, participants indicated having the same level of trust in an anthropomorphic robot as they did in an industrial robot. Trust ratings were generally rather low in both cases [4]. The relationship between users' trust perception and anthropomorphism has also been highlighted in the field of human-automation use or interaction. It has been suggested that anthropomorphism is a critical variable and should be carefully selected while designing human-agent systems [5] Similarly, one of the studies observed that participants significantly trusted a machine more with the anthropomorphic condition in comparison to the mere *Agentic* condition [6]. In summary, we find limited research on understanding the affect of anthropomorphism on trust in HRI and recognize the need for investigating this affect during future studies.

Users' cognitive load has also been identified as one of the significant factors affecting users' trust perception during Human-Machine Interaction (HMI). It has been well-understood that both the trust and cognitive load of the user contributes towards mediating user behaviour during HMI [7]. This suggests that, if a user feels stressed in situations requiring a higher cognitive load, this may result in a degradation of their task performance and may also lower their trust perception. In addition, it has also been showed in HRI literature that participants' subjective rating of cognitive load decreases

as the autonomy increases during teleoperated robot scenarios [8]. As one of the long-term goals in HRI is to facilitate smoother HRI in various environments, and it is expected that some environments may demand higher cognitive load, we are interested in developing robotic technology that could adapt to lower a user's cognitive load and consequently enhance the user's trust in the system. Despite the significance of the relationship between cognitive load and trust perception during HMI, we, to the best of our knowledge, find very little research on investigating the relationship between user's trust and cognitive load in HRI and believe that the current research can crucially contribute to closing this research gap. We, therefore, keeping these aforementioned factors in mind, present a study to investigate the relationship between the concepts of Trust, Cognitive load and Anthropomorphism. In particular, our motivation is to understand how robots can gain trust from humans to ensure a smooth interaction and this may result in reducing cognitive load during HRI.

To achieve our goals, we programmed a "Matching the Pairs" game and adapted it to the Husky and the Pepper robots. The rationale for choosing the Husky and the Pepper was to compare non-humanoid-like and humanoid-like robots. The goal is to enable both robots to play the game as teammates with human users. More specifically, we are interested in exploring the concept of teammates in relation to the HRI during the "Matching the Pairs" game. We intend to determine whether humans trust the robot to undertake its assigned tasks, whether the robot can assist humans in achieving their goal and whether the establishment of a Human-Robot team is possible. Additionally, we also intend to understand the relationship between cognitive load and trust during such robotic interactions. We achieve this through examining whether the participants trust the robot as a teammate in guiding them and facilitating in the selection process to achieve the overall goal of the game (finding the other matching pair). To induce cognitive load, subjects are expected to use their memory to recall positions of the matching pair, while at the same time communicating with the robot and striving for either a good score or just to complete the "Matching the Pair" game. This setup was considered to induce high cognitive load for the participants, thus, making it a good testing ground to investigate the relationship between the cognitive load and trust perception of the users.

II. Background

A. Trust and Factors Affecting Trust in HRI

In general, trust is considered to be a crucial aspect for a team functioning and collaborating successfully. Each member of the team must establish trust, apply it in their work, share it and maintain it. As humans are expected to collaborate with robots in various settings in the future, trust is also one of the critical measures to ensure successful HRI during various social settings. For instance, Hancock et al. (2011) identified trust as an important factor to consider, as the presence or absence of trust directly affects the outcome of HRI [3].

As discussed above, trust in HRI is a complex concept dependent on a number of factors. These factors are divided into three different categories: robot-related, human-related, and environmental characteristics. Robot-related factors including attributes such as anthropomorphism, robot predictability, and robot personality, have a large part to play in the success of the HRI. Similarly, human-related characteristics play a major role in the capability of the human to trust the robot [1], [9]. For example, a human's understanding of the capabilities of the robot may have been shaped by the portrayals of robots in TV and film; this, in turn, would result in either an exaggerated or understated expectation of the capabilities of the robot. Lastly, environmental factors affecting trust would be if a human is trained in close proximity to a robot, it encourages trust in the robot. More specifically, Lee & See (2004) determined a trust importance scale based on the type of environment; trust is less important in a fixed, well-structured environment, for example, procedural based hierarchical organizations in which the order and stability decrease the uncertainty [10]. In contrast, trust is important within dynamic environments, for example, the interaction game for this paper "Matching the Pairs" where the game environment is dynamic, and the participant is uncertain of which marker to choose to make a match. In addition to the aforementioned factors, researchers have observed several other factors including effects of culture, the type of task assigned, task complexity, the provision of training, and the amount of risk involved in the task and in the interaction between human and robot [3].

Of the three antecedents of trust proposed by [3], robot characteristics were acknowledged as the most important factors to consider in trust development. Environmental characteristics provide a neutral outcome on trust, whilst there is some debate on the degree of importance of human characteristics on trust [3]. Consequently, in this paper, we focus on the aspects of robot-related characteristics and their impact on trust in HRI.

B. Trust and Cognitive Load

Cognitive load refers to the amount of effort placed on the working memory during a task. According to Sweller [11], there are three different types of cognitive load produced during problem-solving or learning: 1) intrinsic load, 2) the extraneous load and 3) the germane load. Intrinsic load depends on the complexity of the structure of the material and its association with the learner, the extraneous load is caused by the way this material is presented to the learner, while the germane load is a result of the learner's ability to assimilate the material [11].

In the past, researchers have observed that both trust and cognitive load are related to each other. They have observed that as cognitive load increases, it will negatively affect human's trust perception [7]. As highlighted, we find limited research on the investigation of the relationship of trust and cognitive load in the HRI domain, therefore, we try to understand this relationship from the literature on human-automation/machine use. Research conducted in Human-Automation Interaction suggests that users prefer to

work with the pre-existing trusted system instead of a new system under higher cognitive load [12], [13]. In particular, Biros et al. [13] conducted a study to investigate the variation of human trust in the automated system during a situation involving higher cognitive load. Their results showed that under high cognitive load situations, humans continue to rely on the existing system, although they have less trust in the system. In another study, it was also found that humans show an enhanced level of trust in the system that maintains its reliability and dependability [14].

It has also been shown in one of the studies that humans prefer to use automation during increased workload situations [15]. On the contrary, one study also found that humans trust perception measured subjectively about automation decreases during higher workload situations as they prefer manual work over automation in such situations [16]. However, it has been argued that trust measured subjectively on the automation use is not a correct reflection of the relationship between trust and cognitive load [15]. Additionally, this may have resulted due to an automation bias of the participants. Conclusively, most of the past research suggests that the decrease in the amount of humans' trust perception of the automated system may increase the level of humans' cognitive load. Also both trust and cognitive load are closely related and they affect human behaviour. More specifically, if humans trust, they may stop weighing up pros and cons of working with robots, thus, freeing up cognitive load. To summarize, we believe that limited exploration of the relationship between trust and cognitive load in HRI and findings from the Human-Automation Interaction studies call for the deeper investigation of the effect of cognitive load and trust during HRI. Therefore, in this paper, we try to explore the relationship between cognitive load and trust during the Human-Robot game ("Matching the Pairs") interaction.

C. Measuring Cognitive Load and Trust

Past research has identified a number of factors observable as a consequence of high cognitive load. These factors were commonly based on the human's linguistic behaviour. These behaviors included: enhanced use of pausing, hesitations, and self-corrections [17]–[20], enhanced use of negative emotions, decreased use of positive emotions and several other indicators [21], [22]. We, on the contrary, in this paper, present an approach called "Pupillometry" to measure cognitive load during HRI.

Pupillometry is used as a term in psychology and neurology to describe the act of measuring the diameter of a persons pupil diameters. It has been established that changes in one's eyes' pupil diameter is an indicator of cognitive activity [23], [24]. For instance, German neurologist Bumke had already recognized at the beginning of the previous century that every intellectual or physical activity translates into pupil enlargement [24]. Hess and Polt (1960) achieved a major milestone for pupillometry by discovering that showing semi-nude photos of adults to subjects of the opposite sex would cause their pupils to dilate twenty percent on average [23]. This study provided concrete proof that emotional stimulation causes enlargement of pupil diameter. This notion was later expanded to include other cognitive processes such as memory and problem-solving. Beatty and Kahneman (1966) showed that storing an increasing number of digits in one's memory would cause pupillary dilation [25], while it was also shown by Hess and Polt (1964) that pupil size corresponds with the difficulty of a cognitive task [26].

More recently Just and Carpenter (1993) showcased that pupil responses can be an indicator of the effort to comprehend and process information. They conducted an experiment where participants were given two sentences of different complexities to read while they would measure their pupil diameters. Pupillary dilation was larger while readers processed the sentence that was deemed to be more complicated and more subtle while they read the simpler one [27]. We believe that these findings make the connection between pupillometry and cognitive load clear, as they demonstrate that changes in the properties of an element to be processed (e.g. changing the complexity of a sentence in turn affecting the amount of intrinsic load it will impose on the reader) causes different pupillary responses. Therefore, in this paper, we present pupillometry as a measure of cognitive load during HRI.

To measure pupillometry during HRI, we used Tobii Glasses Pro 2 eye tracking glasses[1], to estimate amounts of cognitive load experienced by the user.

We used a Godspeed questionnaire [28], containing additional questions related to the experiment to measure trust. The additional questions were our primary measure of trust and the Godspeed was used to compare how participants felt about the robot before and after exposure. It is important to note that all Godspeed items were used but only items on trust were analyzed to study the relationship between trust and cognitive load.

III. System Description

Our system comprised of a "Matching the pair game" and we enabled both the Husky and the Pepper to play the game with the users through playing the role of a teammate. The concept is to explore the element of trust in the robot shown by the user during the game. We also measured and recorded the level of human cognitive loads. This was done to understand the relationship between users' cognitive load and their trust perception.

A. Matching the Pair Game

We created an interactive game "Matching the Pairs" as a collaborative task that humans can play together with the robot as a team. The goal of the game was to find all the matching pairs in a set of fiducial markers as shown in Figure 2. Each marker had two sides; the upper side represented the fiducial marker that was used to generate and indicate the marker ID, while the underside indicated the symbol which the marker represented, e.g. a picture of an apple, smiley face

[1]https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/

etc. The rules of the game were simple; we placed 12 markers on the table at the beginning of the game; all of the markers were downwards facing, i.e. the symbols were hidden, and the participant could only see the fiducial marker. The player received 5 points at the beginning of the game. As the game resumed, the player received 1 point for each correct match and received -1 point for each mismatch. The goal was to maintain maximum points while matching all the 12 pairs in order to win the game.

We programmed both the Husky and the Pepper to collaboratively play the Matching the Pair game with the users. Firstly, the robot informed the user about the game rules and the robots specific functionality and reliability (error-rate) while supporting the participant with the game. We created two conditions for both robots through varying the robot's reliability in providing help while giving instructions to the users. The robot provides help as having either a 3% error rate or a 50% error rate and being either human-like or machine-like in appearance to prime the participant's expectations. It is significant to note that results from a pre-test confirmed that husky was deemed machine-like and pepper was perceived more human-like. Similarly, the robot's 3% error-rate was also judged as low error-rate and 50% error-rate was judged as high error rate by the participants before the study.

The game was implemented based on a Finite State Machine (FSM). The algorithm and the rules for the interaction game are described in the activity diagram as shown in figure 1. The user and the robot sat facing each other with twelve-markers placed between them (see Figure3). The user chooses a marker. The marker information was saved including the symbol which it represents with the use of Augmented Reality (AR) functionality; to enable the robot to track the markers. The user later had the following two options regarding the selection of the second marker:

- Ask the robot for help in choosing the other matching pair by saying "help me". In this case, Pepper/Husky recognizes that the participant wants help via the use of the speech recognition engine and responds accordingly. Pepper later responds verbally, while Husky responds visually i.e. displays the results on the screen.
- Choose the marker without asking the robot for help.

B. Online System to measure Cognitive Load

Our system connects the Tobii Pro 2 eye-tracking glasses using the Tobii Pro Glasses 2 python controller [29] library over wifi with the computer (see Figure2). The new system provides information on the cognitive load of the participant at 17 fps. After a very simple calibration phase that takes advantage of the pupillar light reflex [30], the system is able to track the user's cognitive load, based on an empirically set threshold. Our system provides a running average ζ, of pupil diameter:

$$\zeta(d) = \frac{\sum_{t=1}^{N} d_t}{N}$$

where d_t is the current pupil diameter and N the number of frames. It further provides a windowed average:

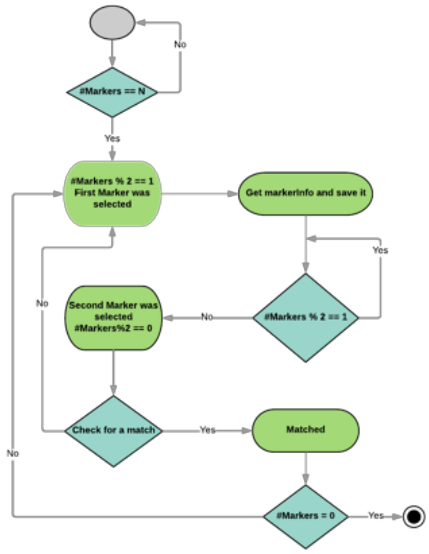


Fig. 1. Activity diagram of the Finite State Machine

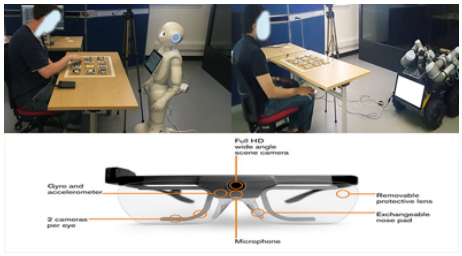


Fig. 2. System setup: The Tobii Glasses 2 are given to each of our participants. Our software records their pupil diameter during the interaction game with the robot. When the cognitive load is perceived as high this can be observed by the experiment as it is displayed on a screen during the interaction.

$$\zeta(d) = \frac{\sum_{t=1}^{15} d_t}{15}$$

where the average is calculated based on the past 15 frames only. Furthermore, our system provides a running peak estimation where we assume that, when the size of the pupil is larger than 70% of the maximum the cognitive load is high.

IV. Research Method

Our research explores two different aspects. Firstly, we focused on investigating the relationship between cognitive

load and human's trust perception during human-robot collaboration. Secondly, we tried to explore the relationship between trust and anthropomorphism. We explored how human's trust perception is affected during the interaction with a humanoid (human-like) and non-humanoid (industrial) robot. Keeping these afore-mentioned aspects in mind, we tried to find answers to the following Research Questions (RQs):

RQ1: Does participants' trust perception decrease due to an increase in the amount of their cognitive load?

RQ1a: Does the relationship between trust and cognitive load depend on the type of robot (humanoid and non-humanoid) along with the low error-rate and the high-rate condition?

RQ2: What is the effect of the type of robot along with error-rate condition on the trust perception and cognitive load of the participants.

RQ2a: Does the humanoid robot reduce the cognitive load more than a non-humanoid robot?

RQ2b: Do humans trust humanoid robots more than non-humanoid robots?

RQ3: How does anthropomorphism (type of the robot: Humanoid vs non-Humanoid) affect participant's trust perception after the interaction, mainly through comparing the pre- and the post-test results?

Our Hypotheses (H) for our RQs are as follow:

H1: Participant's trust perception will decrease with an increase in the amount of cognitive load.

H1a: The relationship between trust and cognitive load will depend on the type of error-rate along with the type of robot.

H2: Both trust and cognitive load of the participants will be affected by the type of robot and the error-rate.

H2a: Participant's cognitive load will be significantly lower for the humanoid robot in the low-error rate condition.

H2b: Participant's trust will be significantly higher for the humanoid robot in the low-rate condition.

H3: Participant's will perceive humanoid robot as more trustworthy compared to the non-humanoid robot after the interaction.

Our hypothesis is based on the finding of the studies reported in Human-Automation literature [12], [13]. Additionally, also on the results of the meta review on the factors affecting trust perception during HRI [3].

A. Participants

We conducted our between-subject study with 26 adults between 25 and 46 years with a mean age of 30.46 and standard deviation of 6.50 interacting with either of the robots in either of 2 modes (high or low error rate). With 5 participants interacting with the Pepper on low error rate, 8 participants interacting with the Pepper high error rate, 5 participants interacting with the Husky low error rate and 8 participants interacting with the Husky high error rate.

It is important to note that our study had 40 participants in total, however, due to difficulties in collecting data for cognitive load through Tobii glasses for some participants, we are reporting results of the 26 adults to understand the relationship between trust and cognitive load. However, for the trust perception mainly comparing the pre- and post-test results, we report results for all the 40 participants as we didn't have missing data for the participants' trust during the pre- and post-test. The conditions were divided as: 10 participants interacting with the Pepper low error rate, 10 participants interacting with the Pepper high error rate, 10 participants interacting with the Husky low error rate and 10 participants interacting with the Husky high error rate.

B. Procedure

We conducted our study in four different steps: Firstly, an information sheet was handed to each participant with information regarding the robot they would be interacting with, along with an image of that robot. A written and approved consent form was obtained from each participant prior to the experiment.

Secondly, each participant was asked to fill in an adaptation of the Godspeed questionnaire [28], containing additional questions related to the experiment on their trust perception of the robot.

Thirdly, each participant played the matching the pairs game with either the Husky or the Pepper robot having one of two different error-rate conditions (3% and 50%) in terms of providing help during the game. It is should be noted that participants were assigned randomly to one of the experimental conditions. Once the game began, both Pepper/Husky indicated the rules of the game, and provided guidance to the participants throughout the game. Low and high error conditions were incorporated on the robots with the occurrence of mistakes 3% (low) and 50% (high) of the time throughout the game. The game finished once all pairs were found.

Lastly, each participant completed the same questionnaire as in the second step. The questionnaires pre- and post were compared using a paired and independent sample t-test.

C. Setup and Materials

We conducted our study in a quiet room that was divided into two parts with a divider as shown in Figure 3. On the one side, the game was placed on the table and either the Husky or the Pepper robot used in each case were placed across the table where the task was performed. The participant was seated in front of the robot. On the other side, another table was placed on which a monitor was situated for the purposes of calibrating the eye tracking equipment. The actions throughout the duration of the task were tracked with the use of AR marker technology and a webcam. Every component of the experiment was controlled from a computer operated from a concealed position behind a blind wall, by the experimenters. It is important to note that we controlled the lighting conditions to maintain the quality of the eye tracking data.

The study had two different phases that were realized in an identical fashion. For the first part, Pepper was used, along with it's voice recognition capabilities for the purposes of the study. In the second part, the non-anthropomorphic robot, Husky was used. Both robots acted as optional assistants for

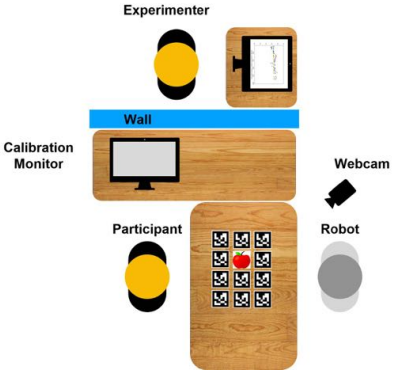


Fig. 3. Overview of the experimental setup.

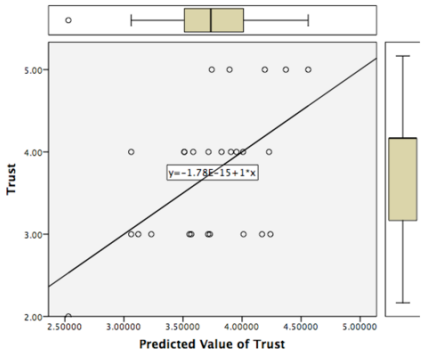


Fig. 4. A linear relationship between trust and the predicted value of trust by our linear regression model.

the participants during the task they were asked to perform. Due to technical difficulties with Google speech recognition, a Wizard-of-oz approach was used with the Husky during the experiments. Nonetheless, the same text based and speech based sentences where produced by the Husky and the Pepper robot. The Husky robot displayed the text on the screen while the Pepper robot uttered the sentences.

D. Measurements

To measure cognitive load during the HRI, we considered the number of peaks per participant as measured by the Tobii Pro 2 eye tracking glasses. In addition, we subjectively measured trust perception on a scale from 1 to 5 through updating Godspeed questionnaires according to our study.

V. Results

To find the relationship between trust and cognitive load or to test H1, a simple linear regression was conducted to predict trust perception based on the number of peaks (maximum cognitive load), error-rate and the type of robot. A significant regression equation was found (F(2,22)=3.395, $p = 0.03$) with an R^2 of .316. The relationship between trust and predicted values of trust is shown in Figure 4. To further analyze H1, we also ran a Pearson correlation between the maximum cognitive load and the trust perception. We find a negative correlation between the two variables, $r = -.409, p = .038$ as also shown in figure 5. This suggests that as the cognitive load increases, the trust perception of the participants decreases. For deeper investigation of the hypothesis and keeping the dynamics of our study, we understand that if users trusted the robot they were playing the game with, they would not be reluctant to ask for assistance or help from the robot. Therefore, we also conducted a Pearson correlation between the amount of questions asked by the participants for help and the maximum cognitive load. It is also interesting to find that there was a negative correlation between the number of questions and the maximum cognitive load, r = -.328, p = .041. This also suggests that in a case when participants were cognitively loaded, they didn't ask for help. Hence, based on the results of all aforementioned tests, our Hypothesis H1 was accepted.

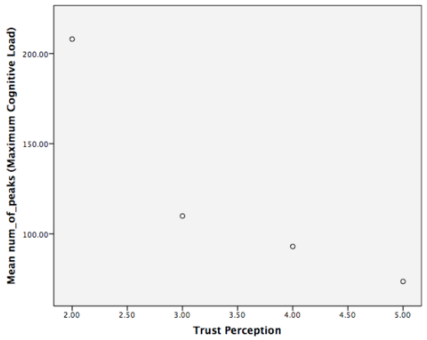


Fig. 5. Scatter Plot highlighting relationship between mean value of maximum Cognitive Load (y-axis) and Trust Perception (x-axis).

To investigate RQ1a and test H1a, we ran a partial correlation analysis on both variables while controlling for the error rates and robot type. We found the following partial negative correlation between the two variables, $r = -.506$, $p = .006$. This also suggested that as cognitive load increased, user perception of trust decreased while controlling for the error rate and the robot type. Hence, H1a was also accepted.

To analyze H2, H2a and H2b, we conducted a multivariate analysis of variance (MANAOVA) with robot type and error-rate as the Independent Variable (IV) and the number of peaks (maximum cognitive load) and trust as the Dependent Variable (DV). We find a statistically significant effect of both the

Variable	Type of Robot, Error-rate	M and SD
Cognitive Load	Husky, low-rate	M: 130.00 , SD: 37.07
	Husky, high-rate	M: 92.85 , SD: 37.07
	Pepper, low-rate	M: 90.60 , SD: 52.04
	Pepper, high-rate	M: 90.62 , SD: 60.89
Trust	Husky, low-rate	M: 3.66 , SD: 1.03
	Husky, high-rate	M: 3.85 , SD: 0.37
	Pepper, low-rate	M: 4.44 , SD: .54
	Pepper, high-rate	M: 3.25 , SD: 0.88

TABLE I
MEAN AND STANDARD DEVIATION VALUES FOR PEPPER AND HUSKY ROBOTS WITH DIFFERENT ERROR RATES.

robot type and error-rate, mainly interaction effect on the trust perception of the participants (F(1,22)=4.833, $p = 0.03$). However, we didn't find a statistically significant effect of the robot type and error-rate, mainly interaction effect on the maximum cognitive load (F(1,22)=.792, $p = 0.383$). This suggests that participants' trust is significantly affected by the type of robot that they are interacting with along with the error-rate of the robot. Hence, our H2 was partially accepted. On the other hand, we didn't find a significant effects of either the type of robot or the type of error-rate individually i.e. main effect on both DVs. However, we witnessed trends through observing the Mean (M) and Standard Deviation (SD) for both the cognitive load and trust. These are shown in Table I. These trends suggest that participants in general were more cognitively loaded in the Husky (non-humanoid) robot condition as compared to the Pepper (humanoid) robot regardless of the error rates. Additionally, participants trusted the Pepper more than the Husky robot in the case of low error rate. Hence, our findings partially did support our H2a and H2b.

To test H3, we also compared the trust perception of the participants through comparing their trust values as reported during the pre- and post-tests through conducting a t-test. We found that participants perceived the Pepper to be significantly more trustworthy after interaction: pre-test (M = 2.8, SD = 0.92) post-test (M = 3.9, SD = 0.99), t(9) = -2.4, p = 0.04. On the contrary, we didn't witness an increase in the trustworthy perception of the Husky robot pre-test (M = 3.1, SD = 1.37), post-test (M = 3.9, SD = 0.994), t(9) = -1.45, p = 0.182. Hence, H3 was also accepted.

It is also interesting to note that participants also perceived the Pepper to be more interactive, lively, social, friendly, and relaxed. Additionally, Pepper was perceived to be significantly more human-like after interaction with: pre-test (M = 2, SD = 0.82) post-test (M = 3.2, SD = 1.14), t(9) = -2.34, p = 0.044. In comparison, before exposure participants perceived Husky as: inert, stagnant, apathetic, and anti-social. Husky was perceived as machine-like before and after interaction with: pre-test (M = 1.8, SD = 1.03) post-test (M = 2.2, SD = 1.23), t(9)= -0.71, p = 0.49. It is to be noted as highlighted earlier that these findings were based on the 40 participants.

VI. DISCUSSION

We see that there is a correlation suggesting that participants were not keen on asking for robot's assistance in the situation demanding higher cognitive load during the game. We understand that this was primarily related to the error-rate condition they were in as we also find this effect as a part of our results. We believe this effect may also be related to the results of the studies performed in the human-automation use [7]. Additionally, there may be a correlation with the age of the participants and the cognitive load that we have not investigated yet. Due to using pupil diameter as a measure for cognitive load, this might be dependent on age, as the reaction time of the pupil changes during aging. Nevertheless we found relatively stable results in the detection of cognitive load in the pupil diameter and therefore we believe it is a good real-time measure for cognitive load. Other measures of cognitive load are of interest for us as well, e.g. verbal features e.g pitch, volume or velocity which we have recently investigated here [20]. Clearly, we need to collect more data to establish that results presented are robust with different participants and that we can exchange task easily. So far our system shows a promising way of detecting cognitive load in HRI, but further evaluation and data collection is needed. Our results also find a correlation between cognitive load and trust while controlling the type of robot. This also refers to further exploration of how anthropomorphising can affect the relationship between the two variables.

We also see that the participants weren't significantly affected in terms of their cognitive load. However, we understand that this is directly related to the number of participants as we did witness a higher mean value for the cognitive load for the Husky robot in comparison with the Pepper robot. This directs and calls for deeper investigation of the relationship of the two variables, in particular, with a higher number of participants. Additionally, we did see that participants trust perception was significantly affected while interacting with either Pepper or Husky possessing different error-rates. We understand that it could be related to the error-condition they were in as participants mean-value for Pepper having a low error rate was significantly higher than the Husky. We didn't find a main effect of either type of robot or the error-rate on either the trust perception or the cognitive load. Once again, we believe that it was due to the limited number of participants and an effect may have been witnessed with a higher number of participants.

Our results based on the t-test conducted to compare participants trust perception after the interaction also showed that anthropomorphising robots' physical appearance along with the communication and increasing the occurrence of human-robot interaction are effective means of increasing human trust levels in their robot counterparts. A positive relationship was established between robot anthropomorphism and trust; the more anthropomorphic the robot, the more it can be trusted. The adoption of facial features on the robot is a technique used for anthropomorphising objects towards human-likeness and subsequently encouraged participants to trust Pepper more than Husky. It is worth noting that the outcome of interacting with a robot is highly dependent on the human involved in the interaction, that is, some participants were utterly reliant

on the robot, others used it as a consultant, whilst others did not depend on the robot at all. This individuality and context specifically emphasizes the demand for further research to be undertaken to continually develop our understanding of HRI.

VII. Conclusions

In this paper, we presented an exploratory study to understand the relationship between human cognitive load, trust and anthropomorphism during a Human-Robot Interaction. To understand the relationship, we created a "Matching the Pair" game that was designed to enable humans to play the game collaboratively with robots. The idea was to understand if humans trust to work with robots as teammates during the game under situations demanding high cognitive load to complete a task. Additionally, we also exploited the aspect of anthropomorphism through enabling humans to either interact with a humanoid (Pepper) or a non-humanoid (Husky) robot. We found that there was an inversely proportional relationship between trust and cognitive load, suggesting that as the amount of cognitive load increased in the participants, their level of trust decreased. Additionally, we also found that participant's perceived the Pepper robot to be more trust worthy in comparison with the Husky robot through comparing our pre- and post-test questionnaire results.

VIII. Limitation and Future Work

It is to be noted that our online system to compute number of peaks (maximum cognitive load) is limited to the specific lighting conditions. We were limited to the number of participants for this study. In particular, we didn't find enough participants to run analysis to account of age as the reaction time of the pupil varies during aging. Therefore, in the future, we intend to further understand the relationship between trust and cognitive load through conducting similar studies with a higher number participants.

References

[1] P. Hancock, D. Billings, and K. Schaefer, "Can you trust your robot?" *Ergonomics in Design*, vol. 19, no. 3, pp. 24–29, 2011.

[2] M. Ahmad, O. Mubin, and J. Orlando, "A systematic review of adaptivity in human-robot interaction," *Multimodal Technologies and Interaction*, vol. 1, no. 3, p. 14, 2017.

[3] P. A. Hancock, D. R. Billings, K. E. Schaefer, J. Y. Chen, E. J. De Visser, and R. Parasuraman, "A meta-analysis of factors affecting trust in human-robot interaction," *Human Factors*, vol. 53, no. 5, pp. 517–527, 2011.

[4] J. Bernotat and F. Eyssel, "Can (t) wait to have a robot at home?-japanese and german users' attitudes toward service robots in smart homes," in *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2018, pp. 15–22.

[5] E. J. de Visser, S. S. Monfort, R. McKendrick, M. A. Smith, P. E. McKnight, F. Krueger, and R. Parasuraman, "Almost human: Anthropomorphism increases trust resilience in cognitive agents." *Journal of Experimental Psychology: Applied*, vol. 22, no. 3, p. 331, 2016.

[6] A. Waytz, J. Heafner, and N. Epley, "The mind in the machine: Anthropomorphism increases trust in an autonomous vehicle," *Journal of Experimental Social Psychology*, vol. 52, pp. 113–117, 2014.

[7] F. Chen, J. Zhou, Y. Wang, K. Yu, S. Z. Arshad, A. Khawaji, and D. Conway, "Trust and cognitive load," in *Robust Multimodal Cognitive Load Measurement*. Springer, 2016, pp. 195–214.

[8] J. V. Draper and L. M. Blair, "Workload, flow, and telepresence during teleoperation," Oak Ridge National Lab., Tech. Rep., 1996.

[9] K. Schaefer, "The perception and measurement of human-robot trust," 2013.

[10] J. D. Lee and K. A. See, "Trust in automation: Designing for appropriate reliance," *Human factors*, vol. 46, no. 1, pp. 50–80, 2004.

[11] J. Sweller, "Cognitive load theory," in *Psychology of learning and motivation*. Elsevier, 2011, vol. 55, pp. 37–76.

[12] S. Oviatt, R. Coulston, and R. Lunsford, "When do we interact multimodally?: cognitive load and multimodal communication patterns," in *Proceedings of the 6th international conference on Multimodal interfaces*. ACM, 2004, pp. 129–136.

[13] D. P. Biros, M. Daly, and G. Gunsch, "The influence of task load and automation trust on deception detection," *Group Decision and Negotiation*, vol. 13, no. 2, pp. 173–189, 2004.

[14] R. Parasuraman and C. A. Miller, "Trust and etiquette in high-criticality automated systems," *Communications of the ACM*, vol. 47, no. 4, pp. 51–55, 2004.

[15] R. Parasuraman and V. Riley, "Humans and automation: Use, misuse, disuse, abuse," *Human factors*, vol. 39, no. 2, pp. 230–253, 1997.

[16] H. A. Ruff, S. Narayanan, and M. H. Draper, "Human interaction with levels of automation and decision-aid fidelity in the supervisory control of multiple simulated unmanned air vehicles," *Presence: Teleoperators & Virtual Environments*, vol. 11, no. 4, pp. 335–351, 2002.

[17] A. Berthold and A. Jameson, "Interpreting symptoms of cognitive load in speech input," in *UM99 User Modeling*. Springer, 1999, pp. 235–244.

[18] A. Jameson, J. Kiefer, C. Müller, B. Großmann-Hutter, F. Wittig, and R. Rummer, "Assessment of a users time pressure and cognitive load on the basis of features of speech," in *Resource-adaptive cognitive processes*. Springer, 2010, pp. 171–204.

[19] M. A. Khawaja, N. Ruiz, and F. Chen, "Think before you talk: An empirical study of relationship between speech pauses and cognitive load," in *Proceedings of the 20th Australasian conference on computer-human interaction: Designing for habitus and habitat*. ACM, 2008, pp. 335–338.

[20] J. Lopes, K. Lohan, and H. Hastie, "Symptoms of cognitive load in interactions with a dialogue system," in *Proceedings of the Workshop on Modeling Cognitive Processes from Multimodal Data*. ACM, 2018, p. 4.

[21] M. A. Khawaja, F. Chen, and N. Marcus, "Using language complexity to measure cognitive load for adaptive interaction design," in *Proceedings of the 15th international conference on Intelligent user interfaces*. ACM, 2010, pp. 333–336.

[22] ——, "Analysis of collaborative communication for linguistic cues of cognitive load," *Human factors*, vol. 54, no. 4, pp. 518–529, 2012.

[23] E. H. Hess and J. M. Polt, "Pupil size as related to interest value of visual stimuli," *Science*, vol. 132, no. 3423, pp. 349–350, 1960.

[24] E. H. Hess, *The tell-tale eye: How your eyes reveal hidden thoughts and emotions*. Van Nostrand Reinhold, 1975.

[25] J. Beatty and D. Kahneman, "Pupillary changes in two memory tasks," *Psychonomic Science*, vol. 5, no. 10, pp. 371–372, 1966.

[26] E. H. Hess and J. M. Polt, "Pupil size in relation to mental activity during simple problem-solving," *Science*, vol. 143, no. 3611, pp. 1190–1192, 1964.

[27] M. A. Just and P. A. Carpenter, "The intensity dimension of thought: pupillometric indices of sentence processing." *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, vol. 47, no. 2, p. 310, 1993.

[28] C. Bartneck, D. Kulić, E. Croft, and S. Zoghbi, "Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots," *International journal of social robotics*, vol. 1, no. 1, pp. 71–81, 2009.

[29] D. D. Tommaso, "Tobii pro glasses 2 python controller," Webpage, Jul. 2018. [Online]. Available: https://github.com/ddetommaso/TobiiProGlasses2$_{PyCtrl}$

[30] A. L. Kun, O. Palinko, and I. Razumenić, "Exploring the effects of size and luminance of visual targets on the pupillary light reflex," in *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. ACM, 2012, pp. 183–186.