UNIVERSIDADE D
COIMBRA

João Miguel Gomes Pereira

Monitoring of Ocular Data for Robotic Control

Dissertação no âmbito do Mestrado Integrado em Engenharia Electrotécnica e de Computadores, especialização em Automação orientada pelo Professor Doutor Rui Duarte Cortesão e apresentada à Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Abril de 2019

UNIVERSIDADE D
COIMBRA

Faculty of Science and Technology
of the University of Coimbra

MONITORING OF OCULAR DATA FOR ROBOTIC CONTROL

João Miguel Gomes Pereira

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Electrical and Computer Engineering, supervised by Rui Pedro Duarte Cortesão and presented to the Faculty of Science and Technology of the University of Coimbra.

April of 2019

UNIVERSIDADE D
COIMBRA

AGRADECIMENTOS

Em primeiro lugar começo por agradecer ao meu orientador, Professor Rui Pedro Duarte Cortesão, sem a sua confiança e paciência não teria sido possível realizar esta dissertação. Agradeço também ao Instituto de Robótica de Coimbra, por me proporcionar as condições necessárias para a realização deste trabalho, e a todos os professores do departamento por me providenciarem os seus conhecimentos.

Agradeço também a todos os meus colegas da Universidade de Coimbra que presenciaram esta jornada, em especial, Dylan Bicho, aos meus colegas de laboratório, Hélio Ochoa, Rui Fernandes, Miguel Mendes, Diogo Vaz e Luís Santos pelo aconselhamentos fornecidos, e aos meu colegas da Residência, Luís Machado, João Amado, António Vieira e Daniel Torres.

Em especial nota, agradaço aos meus Pais pelo apoio emocional e pelo espírito crítico e curiosidade que me embutiram ao longo dos anos, sem o seu esforço e dedicação não teria chegado ao final desta meta. Agradeço também ao meu irmão por todo o apoio que me deu e pela sua paciência ao longo dos anos.

João Miguel G. Pereira

Coimbra, Abril de 2019

RESUMO

Esta dissertação tem como objetivo o controlo de um braço robótico, Kinova Jaco^2, utilizando movimentos oculares de um utilizador. Esse tipo de controlo é explorado para fins assistivos, utilizadores com problemas de mobilidade ou simplesmente funcionalidades extra para interação homem-máquina.

Uma breve análise do *Eye Tracker* é apresentada, onde são especificados os requisitos para sua comunicação com a interface de controle do robô escolhido. O uso da interface ROS permite o desenvolvimento dos componentes necessários para serem genéricos e modulares, que podem ser usados com outros braços robóticos. Para realizar o controle do robô considera-se a existência dos controladores ROS já desenvolvidos para o robô escolhido, assim os únicos requisitos são a geração de poses cartesianas desejadas.

Usando dois *blinks* para sinalizar a geração de uma nova pose desejada, podendo esta ser gerada a partir da interseção do olhar do utilizador com a *point cloud*. Para isso, é necessário o conhecimento da localização do *Eye Tracker* no mundo obtido através de uma *rigid body transform* com marcadores *Aruco* colocados no mundo. A análise das características locais do ponto de interseção entre o olhar e a *point cloud* permite a extração da nova pose desejada. Um método de geração de pose adicional é implementado onde a seleção dos marcadores usando o foco visual do utilizador, sendo que para cada marcador a predefinição de uma pose desejada é necessária, o que permite o controlo do robô considerando a preparação do espaço de trabalho à priori.

Em conclusão, os métodos desenvolvidos alcançaram uma precisão significativa, tendo em conta os erros inerentes presentes nos dados extraídos do Eye Tracker. Os métodos desenvolvidos para a geração de poses são específicos à aplicação, permitindo ao utilizador realizar tarefas semi-predefinidas mais rapidamente do que usando telemanipulação. Resultando em uma redução de ~30% no tempo de operação do robô.

Palavras-Chave: Kinova Jaco, *Pupil Eye Tracker*, *ROS*, *Aruco*, tele-manipulação

ABSTRACT

This dissertation has as its objective the control of a robotic arm, Kinova Jaco^2, using ocular movements of a user. This type of control is explored in an assistive context, for disabled users or simply as extra functionality in human-machine interaction.

A brief analysis of the Eye Tracker is presented, where the requirements for its communication with the chosen robot control interface are specified. The use of the ROS interface allows the development of the necessary components to be generic and modular, which can be used with other robotic arms. To accomplish the robot control it is considered the existence of the ROS controllers already developed for the chosen robot, as such the only requirements are the generation of cartesian desired poses.

Using double blinks to hint the generation of a new desired pose, then these can be generated from either the intersection of the detected user gaze with the world point cloud. For which is required the knowledge of the location of the Eye Tracker in the world obtained using a Rigid Body transform on Aruco markers placed in the world. Analysing the local features of the intersection point between the gaze and the point cloud allows the extraction of a new desired pose. An additional pose generation method is implemented where the selection of world markers using the user gaze, for each marker the existence of a predefined desired pose is required, allows for robot control considering preparation of the workspace a priori.

In conclusion, the methods developed achieved significant accuracy, considering the inherent errors present in the data extracted from the Eye Tracker. The developed methods for pose generation are application-specific, these allow the user to accomplish semi-predefined tasks faster than telemanipulation. Resulting in a ~30% reduction in the operation time of the robot.

Keywords: Kinova Jaco, Pupil Eye Tracker, ROS, Aruco, Telemanipulation

CONTENTS

LIST OF ACRONYMS

DIY	Do-It-Yourself
DOF	Degrees Of Freedom
FOV	Field Of View
FPS	Frames Per Second
HMC	Human Machine Cooperation
HMI	Human Machine Interaction
IP	Internet Protocol
IPC	Inter-Process Communication
IR	InfraRed
JSON	JavaScript Object Notation
OS	Operating System
PCA	Principal Component Analysis
PNP	Perspective-N-Point
RGB	Red, Green, Blue
RGB-D	Red, Green, Blue and Depth
ROS	Robot Operating System
STD	STandard Deviation
TCP	Transmission Control Protocol

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Human awareness is mostly a product of the visual system, as this is the dominant human sense used by humans to interact with the real environment. It also has a deep connection to the internal cognitive processes and can be described as unconscious and intent driven. Then from an analysis of this process, especially the visual attention which can be considered a reflection of the internal thought process, results the potential acquisition of user intention.

Visual servoing for robotic applications can be considered an extremely researched area, however most of this research was costly and bulky, only in recent years has the technology advancements allowed the implementation of similar experiments in mobile applications at affordable prices. One such application is the research into Human-Machine Interaction (HMI), where the multi-sensor systems are common. The high dimensionality of the captured information requires high computational performance, especially since this specific application usually requires the aforementioned computation to be done in real-time.

Considering the robotic trend in the industrial setting has become Human-Machine Cooperation (HMC) then, a possible approach is visual servoing driven by human intent, through eye analysis. This data in junction with a number of different human cues, most common include tactile, speaking and ocular data, serve as the basis of this cooperation trend in HMI and are worth exploring.

This type of research instruments also have extremely useful applications in the assistive robotics field which usually requires less accuracy and have as main focus the systems applicability.

1.1 Objectives

This dissertation has as its objective the control of a robotic arm, in this case the Kinova Jaco[2], which is a robot with 6 Degrees Of Freedom (DOF), using ocular movements of a user. To accomplish this task generic modules that output the desired cartesian pose are developed for the desired robot interface, the Robotic Operating System (ROS).

1.2 Contributions

The work presented provides a complete generic ROS environment, including simulation, designed for robot control using the Pupil eye tracker in section (3). This includes the necessary initial gaze processing, world perception and two different methods to generate a robot goal pose. A complete testing of each of the featured methods is also presented. This development also takes future researches into account which adds genericness as a prerequisite.

This dissertation is focused on the initial development of the interface between the chosen eye tracker and the ROS which results in its focus being on the processing of the acquired data rather than robot control algorithms. Although the robot used in the dissertation is the previously stated Kinova Jaco^2, the research environment developed is completely generic and can easily be used with other robots provided the existence of its support for the ROS.

1.3 Outline

To simplify the document structure of this dissertation all the document sections are divided according to the modular components of the developed system, which can be observed in the following diagram (1).

This results in the following document structure:

Chapter 1 - *Introduction*

Serves as an initial introduction to the dissertation, including the objectives, contributions and a brief overview of the document structure.

Chapter 2 - *Background*

A very brief introduction to the research already developed in this scientific area, focusing on those that make use of the same eye tracker as this dissertation.

Chapter 3 - *Pupil Eye Tracker*

Serves as an introduction to the technical capabilities of both the Pupil Software and the Pupil eye tracker cameras.

Chapter 4 - *ROS Pupil Package*

Has both the real and simulated environments complete setup, including the communication between the Pupil Software and the ROS environment and the initial processing required for the data extracted. Includes in-depth analysis of the expected quality of the data generated from each of the individual components developed.

Chapter 5 - *ROS Gaze Package*

Development of methods for robot goal pose generation, with a brief individual analysis of the results obtained for each individual module.

Chapter 6 - *Results*

Brief evaluation of the results of the complete system for the methods developed in the previous chapter, using practical demos.

Chapter 7 - *Conclusions*

Final feedback of this dissertation results, and suggestions for future work.

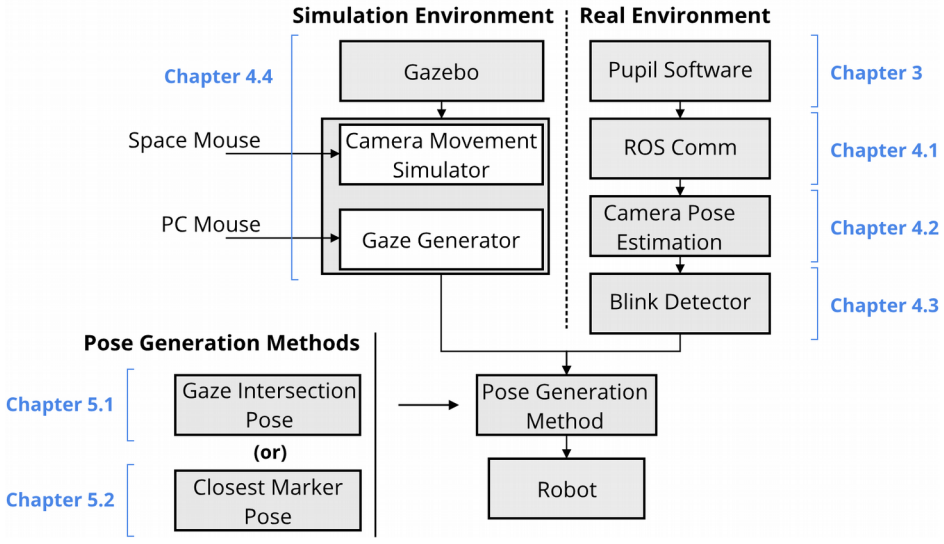


Figure 1: Complete system diagram, with corresponding document section signaled in blue.

As a remark, the chosen chapter names follow the developed ROS modules names, and the ROS platform naming guidelines which names each individual module as a packages.

2 Background

Although the eye tracking research can be traced back more than a century [1], only recently has the technology advancements allowed for easy to use, affordable, and compact solutions to eye tracking. These have recently become viable by using well known computer vision algorithms for video processing of a camera facing the desired eye for which the computers were previously incapable considering the real-time requirements.

These eye tracking devices take advantage of certain particularities of the human eye structure, especially since for full resolution of a particular object this must be projected into the fovea of the eye. This drives the eye movements according to the user visual focus which in turn provides an indication of the the items being observed. It is also important to note that the human eye movements can be divided into two different types, saccades and fixations, which represent sudden and instantaneous eye movements followed by brief fixations, 200-600ms [2]. However these fixations can not be considered completely stable due to small jittery motions, of generally less than one degree, which are imperceptible to the user but are the origin of eye tracking errors and need to be correctly filtered in software. It is also worth to noting that humans visual acuity is at its peak in the foveola region of the eye, however since this area represents approximately 1º degree of Field Of View (FOV), then it expected that the overall accuracy of any eye tracking system will be limited by this factor.

These types of systems provide extremely useful information since eye gaze reveals user intent [3] [4] [5] and can be used to infer the user cognitive load [6] which results in high applicability in psychological or analytic studies [7] [8] [9] [10] for a diverse of applications. Only recently has the technology allow for sufficient accuracy and reduced latency for these to be useful in robotics, especially in an assistive capacity and/or alternatives to normal robot operation. These take an indirect approach to robot control by either creating methods for shared autonomy, making use of the extra information to resolve the extra dimensionality presented in a 7 DOF robot [11], creating simple support systems to increase the practicality of low dimensional joysticks in higher dimensional

robots [12], using the user inferred attention point for error detection during robot operation [13], or by creating control schemes to combine user gaze and robot manipulator inputs into a single robot goal [5] [14] [15].

Another area of active research where these types of head-mounted systems are beneficial is in AR and VR environments [16] where the estimated user focus point is especially useful to further refine user interfaces.

Currently there are two main methods for eye detection, either head-mounted trackers with cameras directed to both the eye and world scenes, two cameras are always required in this case so calibration is possible, or static world cameras that use head detection to estimate the eye locations and subsequently its gaze orientation. However, for the latter case, the accuracy depends highly on the quality of the cameras used and limits the user movements to a predefined world region which in our case is undesired.

Considering, the affordability and compactness as prerequisites then the resulting head-mounted eye trackers available are reduced to the Tobii Pro Glasses 2 [17] and Pupil Labs Eye Tracker [18]. These have similar capabilities and approaches to gaze extrapolation and both achieve acceptable accuracy [19]. Mean overall accuracy of 0.84º for the Pupil Eye Tracker and 1.42º for the Tobii Pro Glasses. However, it is important to note that these are highly dependent of the quality of the initial calibration procedure used in each eye tracker [20]. Furthermore, considering that pupil detection errors propagate throughout the subsequent system layers, then the obvious choice for this dissertation becomes the eye tracker with the highest overall accuracy and most versatile calibration procedures [19] [20], the Pupil eye tracker.

3 Pupil Eye Tracker

3.1 Overview

The Pupil eye tracker, or just Pupil, is a wearable mobile eye tracking headset, developed by Pupil Labs [18], from Berlin, with one world camera and one Infrared (IR) eye camera for dark pupil detection. Since its initial development a RGB-D world camera, Intel Realsense R200 [21], and a second optional eye camera were also introduced. The cameras communicate with a computer using USB 3.0, where the camera video streams are read using the open-source Pupil Capture software for real-time pupil detection, gaze mapping, recording, and other functions.

The development focus for the Pupil was affordability and extensibility, as such, both the hardware and software take a modular approach to its design and most of its components can be bought inexpensively. As such, a Do-It-Yourself (DIY) tutorial for the hardware development and the CADs for the mounting frames are also provided by Pupil Labs resulting in a flexible research environment. Considering the modular approach the final product can be divided into three major modules, the world and eye cameras, and the mounting frame.

Figure 2: Photo of the Pupil Labs Eye Tracker used in the present dissertation, monocular version with the Intel Realsense R200 World Camera.

Since the frame structure was developed with additive fabrication methods in mind, and its CAD is provided, the generic mounting frame provided can be easily swapped with a custom made. Facilitating the application of this eye tracker for non-generic user cases such as users with debilitating medical conditions or simply to account for head growth in children.

3.2 Analysis of the Cameras Capabilities

3.2.1 Eye Camera

As described, the pupil eye tracker has one world camera and one or two eye cameras. The current version of the eye camera has the following principal characteristics:

- Global Shutter – Improves pupil detection since the movements, saccades are extremely fast;
- Fast FPS – 2 modes, 200 FPS at 200x200 resolution or 120FPS at 400x400.

Figure 3: Pupil Eye Camera with the IR led visible.

Considering that the most useful information pertaining this dissertation will be the eye blinking and fixation points in time. Since eye blinks take approximately 100ms and fixations require an acceptable period of time, then the 120 Frames Per Second (FPS) mode with higher resolution has sufficient update frequency being the chosen mode in subsequent work.

It is also important to notice that the eye camera uses a surface mount IR led to illuminate the user's eye reducing the effect of visual noise from ambient light sources. Although experiments relating the actual Pupil eye tracker to eye damage have not been made, the pupil hardware has been certified for IR safety according to EN62471:2008.

3.2.2 World Camera

As the world camera, the Pupil makes use of the Intel Realsense R200 RGB-D camera. This module has 3 cameras, a RGB camera with rolling shutter and 2 displaced IR cameras which are used in junction with an IR laser projector to estimate the depth information.

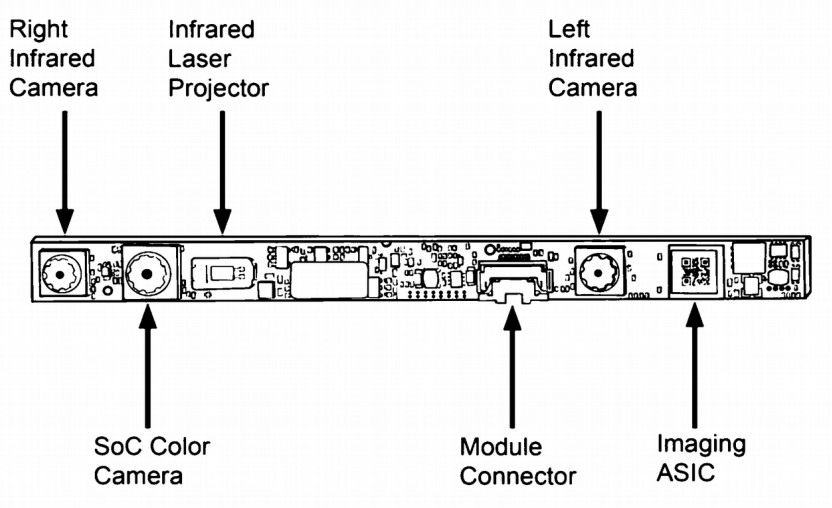


Figure 4: Intel Realsense R200 RGB-D camera module diagram.

Since most of this dissertation will be influenced by the data obtained from this camera, especially the depth stream it produces then it is beneficial to measure its quality. To achieve this several frames of an orthogonal wall to the camera are acquired at different distances resulting in average and standard deviation images for each distance. For display purposes these are then projected to a top view, as seen in figure 5. The plane is verified, previously to the data collection, to be orthogonal using a plane fitting algorithm from which the camera is oriented as to obtain the best orthogonality results at each distance.

Important to note that depth values lower than 700mm or higher than 3500mm are considered invalid and do not influence neither the average nor the standard deviation.

Table 1: Depth values of a orthogonal plane to the camera, averaged from 100 frames, in millimetres.

	750mm	**1250mm**	**1750mm**	**2250mm**	**2750mm**
Average (mm)	740.6	1268.5	1784.1	2303.3	2838.9
STD (mm)	2.0	2.8	6.9	11.6	29.3
Average Error (mm)	-9.4	18.5	34.1	53.1	88.9

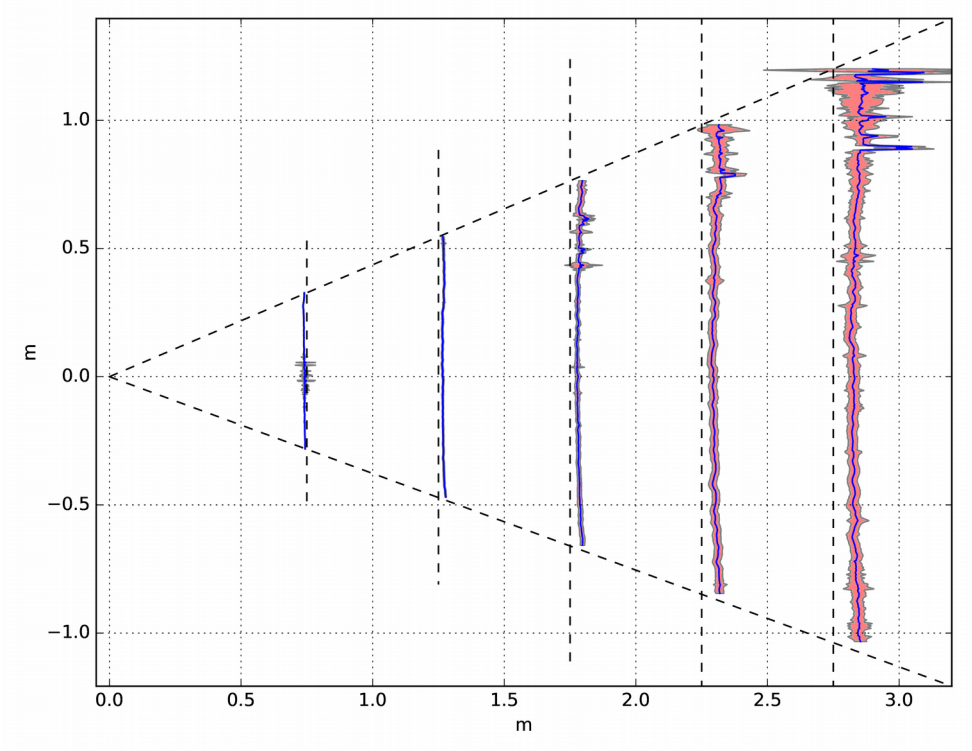


Figure 5: Depth image averages and standard deviations projected into the X-Z plane of the depth camera optical coordinate system.

The RGB-D world camera option has now been discontinued by Pupil Labs. The only option currently available is a normal 30 FPS at 1080p resolution RGB camera. Seeing that the camera used in this dissertation is the discontinued version, then it is useful to present similar RGB-D camera products, that may be implemented as a substitute for the R200 camera, acknowledging the advantages that such upgrades would bring to the present dissertation.

The following products are all Intel Realsense Products and are considered Class 1 Laser Products.

Table 2: Comparison between available Intel Realsense RGB-D cameras

	R200	**D415**	**D435**
Environment	Indoor and Outdoor	Indoor and Outdoor	Indoor and Outdoor
Depth Technology	Active IR Stereo	Active IR Stereo	Active IR Stereo
Sensor Technology	Rolling Shutter	Rolling Shutter	Global Shutter
Depth Resolution	Up to 480p	Up to 720p	Up to 720p
Depth FOV (HxV)	56ºx43º	63.4ºx40.4º	85.2ºx58º
Minimum Depth	0.32 m	0.16 m	0.11 m
Maximum Depth	Up to 10m	Up to 10m	Up to 10m
RGB Resolution	Up to 1080p	Up to 1080p	Up to 1080p

RGB FOV (HxV)	70x43º	69.4ºx42.5º	69.4ºx42.5º
RGB Frame Rate	30 FPS at 1080p	30 FPS at 1080p	30 FPS at 1080p
Dimensions (WxHxD)	101x10x4mm	99x20x23mm	90x25x25mm
Price	79$	149$	179$

The use of the R200 Camera is no longer advised[8] since it has become a legacy product with limited hardware and software support. The D415 camera currently in production with full hardware and software support is a good option for most applications, the most concerning issue becomes the rolling shutter that just like in the R200 camera can be a problem in mobile applications, which is the case. Finally the D435 camera effectively removes the previous cameras issues having a global shutter and a better FOV. The only drawback is its bigger size.

3.3 Pupil Software

The pupil software, written mostly in Python 3, can be divided into two major components, the Pupil Capture which is the main focus for this dissertation and the Pupil Player which can be used to visualize previously recorded data from Pupil Capture. Both have similar structure, using individual plugins joint together to form the actual software. Since this type of software implementation has distributed design then it requires a internal communication server, which in the Pupil Software case is built around the ZeroMQ library which is called the Inter Process Communication (IPC) backbone of the application. This server uses 2 Transmission Control Protocol (TCP) sockets with random Internet Protocol (IP) addresses, which are known to all the plugins spawn by the application, one used as a one to many publisher that the spawned plugins can use to publish whatever messages are desired, and another as a many to one subscriber. These can be used by to read/receive information between the different plugins in a N-to-N design. This messages are required to be a multi-part, using the first part as a message name/id and the second as the actual payload. This lets the developers parse the payloads according to the paired topic name. Internally the Pupil Software payloads are serialized and compressed using the MsgPack library, resulting in a similar message structure to JavaScript Object Notation (JSON).

8 The R200 camera uses deprecated api, for its correct function in linux environment it is required the use of the linux 16.04 with a specific patched kernel, 4.4.0.

Since the ZeroMQ library creates a computer-wide server then, as long as publisher and subscriber addresses are known, an external application can communicate with Pupil Software. Since these addresses are created randomly then its required to use the Pupil Remote plugin, through which an external source can request the IPC backbone addresses. It is also important to notice that the actual camera images are not published into the IPC backbone by default, the use of the Frame Publisher plugin is required in junction with code changes to the Pupil Software so that the three frames, Eye RGB, World RGB and World Depth are accessible externally.

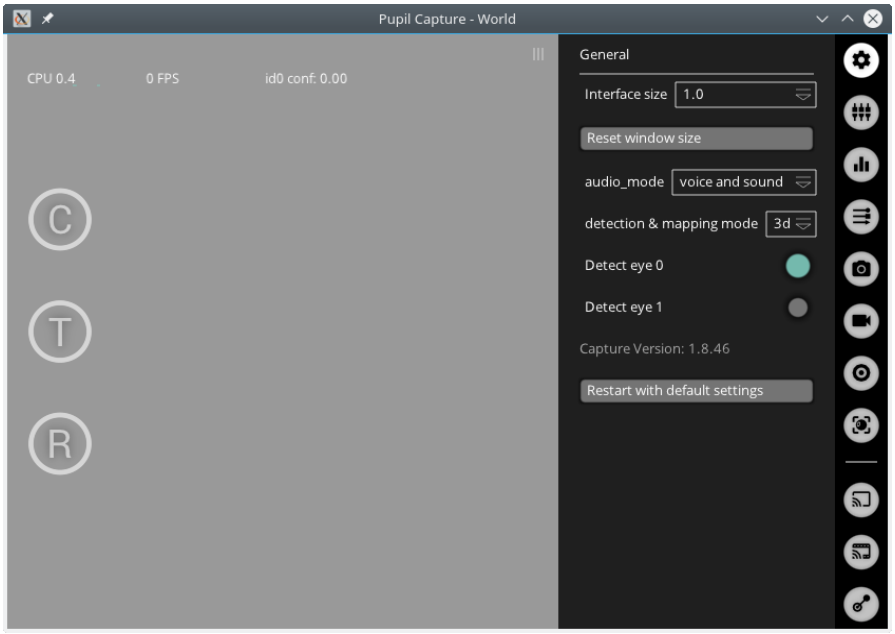


Figure 6: Pupil Software GUI Design

The Pupil Software principal features are the real-time pupil detection and the calibration algorithms, which uses the detected pupil locations in relation to several screen markers to estimate the parameters of a two bivariate polynomial of adjustable degree that maps the detected pupil positions in the eye camera into the world scene. There are also other calibration methods implemented including manual marker calibration, natural features calibration and recent updates introduced a tip of finger calibration. In this dissertation the chosen calibration methods is the screen marker calibration, since access to a computer is not limited. All of the aforementioned algorithms have as its basis a bundle adjustment algorithm to estimate the unknown parameters.

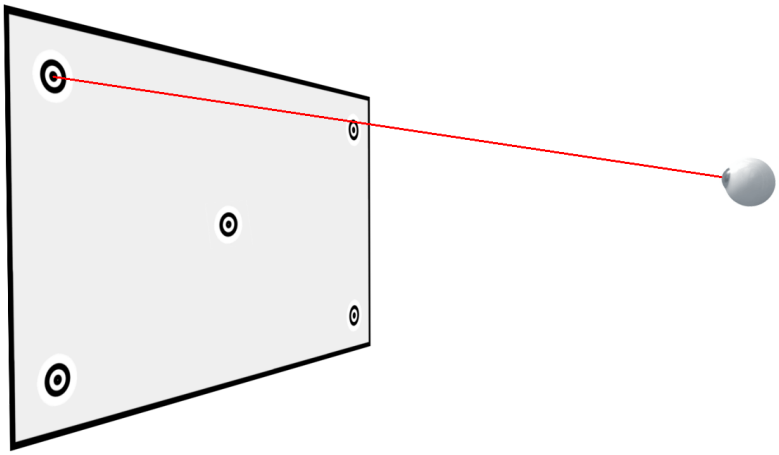


Figure 7: Screen calibration with 5 markers, used in Pupil Software.

Important to notice that since the calibration adjusts parameters that relate the eye position to the world camera coordinate system, whenever the user moves or adjusts physically the Pupil headset a new calibration is required. This also means that physical intensive movements from the user are not advised since the probability of a small shift in the headset position can happen resulting in the incorrect gaze vector from that point onwards.

3.3.1 Pupil Detection Algorithm

The pupil detection algorithm implemented in the Pupil Software, especially designed for dark pupil in IR illuminated eye images, is composed by several minor algorithms. Starting with conversion to a grayscale image, initial region estimation using strongest response for centre surrounded features. Edges detection using Canny algorithm and edge filtering based on neighbouring pixel intensity excluding edges originating from spectral reflections. Finally, edges are filtered using connected components based on curvature continuity and pupil candidates are formed with a ellipse fitting algorithm.

Although the pupil ellipse is detected, and can be used directly using the aforementioned two bivariate polynomial mapping for the 2D gaze vector extrapolation, it is also possible to overlay this pupil detection into a 3D eye model estimate to obtain a 3D gaze vector. This feature implemented by Yuta Itoh and Jason Orlosky makes use of the algorithm in [22] resulting exactly in the required outputs, a 3D gaze vector in the eye camera optical coordinate frame which can be transformed into the world camera

coordinate frame using the estimated extrinsic parameters from the previously mentioned bundle adjustment.

Although the eye detection is fairly accurate, it is important to note that this accuracy is directly related with the calibration quality since the extrinsic parameters that relate the eye camera and world camera frames are estimated during the calibration process. As such it is advised to follow the necessary guidelines for the calibration process as to limit the presence of posterior mapping errors from which the fixation of the user head during the full calibration process is the most important.

It should also be noted that the pupil detection algorithm performance is affected by external IR intensive lights, which slightly reduces the quality during outdoor operation, however since this dissertation is focused in indoor environments this problem is dismissed.

4 ROS Pupil Package

4.1 ROS Communication with Pupil Software

Just as mentioned in previous chapter, the use of the Intel Realsense R200 camera restricts the Operating System (OS) by requiring the Linux version 16.04 with a modified version of the 4.4.0 kernel. Consequently the ROS version becomes limited to the Kinetic version, since the new updates require the minimum Linux version 18.04, as such both the OS and ROS must be downgraded, in addition the python version linked to the ROS is also the downgraded python 2.7 version which is not compatible with python 3, which is the version used in Pupil Software.

Taking in consideration all of the aforementioned requirements, then to create a communication bridge between the ROS and Pupil Software it is necessary to use the ZeroMQ, MsgPack libraries and both the Pupil Remote and Frame Publisher plugins in junction with the modified version of the Pupil Software, allowing external link to the Pupil Software IPC backbone, which provides the developers with access to the internal Pupil Software messages.

To obtain the aforementioned effect, a C++ ROS package was created with its single purpose being the communication between the ROS and Pupil Software. As such this package uses the Pupil Remote plugin to request the internal Pupil Software IPC addresses, subsequently connecting to them using the ZeroMQ library [A]. Considering the previously mentioned message structure for the Pupil Software which is based of the JSON structure, then the messages received from the Pupil IPC backbone need to be translated, as such the use of the coupled message topic name and preceding knowledge of the information sent for each of the message types allows the extraction of the desired information. This de-serialization, just like in Pupil Software is also accomplished using the MsgPack library, specifically the adaptors module of the ZeroMQ Library for which it is necessary to define explicitly the different messages data structure.

Internally, the Pupil Software has 4 types of messages:

- **Log Messages (**Messages used internally for Pupil Software Logging)
- **Notification Messages** (Messages used to coordinate all the individual plugins actions. For example, notifications to spawn any desired plugin, calibration or recording start/stop notifications permit control of previously prepared plugins from external sources)
- **Frame Messages** (Messages with the actual images payloads, with respective size and format specified. Each individual camera has its frame message.)
- **Gaze/Pupil Messages** (Can be considered a single message, since all the Pupil Messages, one for each eye camera available, are actually used internally to produce a single gaze message)

Important to notice, that it is possible to select which of the previously mentioned topics the developer wants to receive when subscribing to the IPC backbone using the ZMQ library. Considering that only the frame of each camera and gaze/pupil messages are required in the ROS environment then it is necessary to create their ROS counterparts. By analysing the previously mentioned messages payloads and considering the 3D gaze mapping as the calibration methods of choice then the resulting message structure becomes:

- **Gaze Datum**, resulting message from the pupil detection algorithm;
  - **topic** - name of the topic subscribed to receive this message;
  - **confidence** - normalized confidence value from the pupil detection algorithm;
  - **norm_pos** - normalized coordinates (u,v) of the eye point in the world image plane, projection from the gaze_point_3d into the image plane;
  - **timestamp** - time of correspondent source image, used for synchronization;
  - **gaze_normal_3d** - 3D vector of the gaze, in junction with eye_center_3d it fully defines the gaze line;
  - **gaze_point_3d** - 3D position of the gaze point in the world camera frame, not really useful since it requires the binocular option, with the monocular version it has fixed depth;
  - **eye_center_3d** - estimated 3D center of the eye in the world camera frame;
  - **base_data** - list of the Pupil Datum messages used to generate the current Gaze Datum message;

- **Pupil Datum**, individual message for each eye, every information is relative to the respective eye camera coordinate systems;
  - **topic** - name of the topic subscribed to obtain this message;
  - **index –** index of the closest world image, in time;
  - **timestamp** - time of the correspondent eye camera source image, used for synchronization;
  - **id** - boolean flag to specify right of left eye;
  - **confidence** - normalized confidence of the pupil detector on the current message information;
  - **method** - Detection method used, in this dissertation is, *3d c++*;
  - **ellipse** - 2D ellipse of the detected pupil in pixels, result from the ellipse fitting of the pupil detection algorithm;
  - **model_birth_timestamp** - initial time of the 3d eye model, useful to verify if the model has completed the eye fitting;
  - **model_confidence** - normalized confidence of the 3d eye model fitting;
  - **model_id** - each new model has a different id;
  - **theta** - polar coordinate of the eye model;
  - **phi** - polar coordinate of the eye model;
  - **circle_3d** - 3D ellipse of the pupil detected in the corresponding source image;
  - **sphere** - resulting sphere generated by the eye model in the eye camera frame;
  - **projected_sphere** - the previous sphere projected into the eye camera image.

Since the gaze and pupil messages in the Pupil Software are constructed from minor types such as *2d_ellipse*, *sphere* and *3d_circle*, then the resulting ROS message structure can be represented by the diagram (8), where it is possible to visualize the different components that constitute the custom created ROS message *pupil_msgs/gaze_datum.msg*.

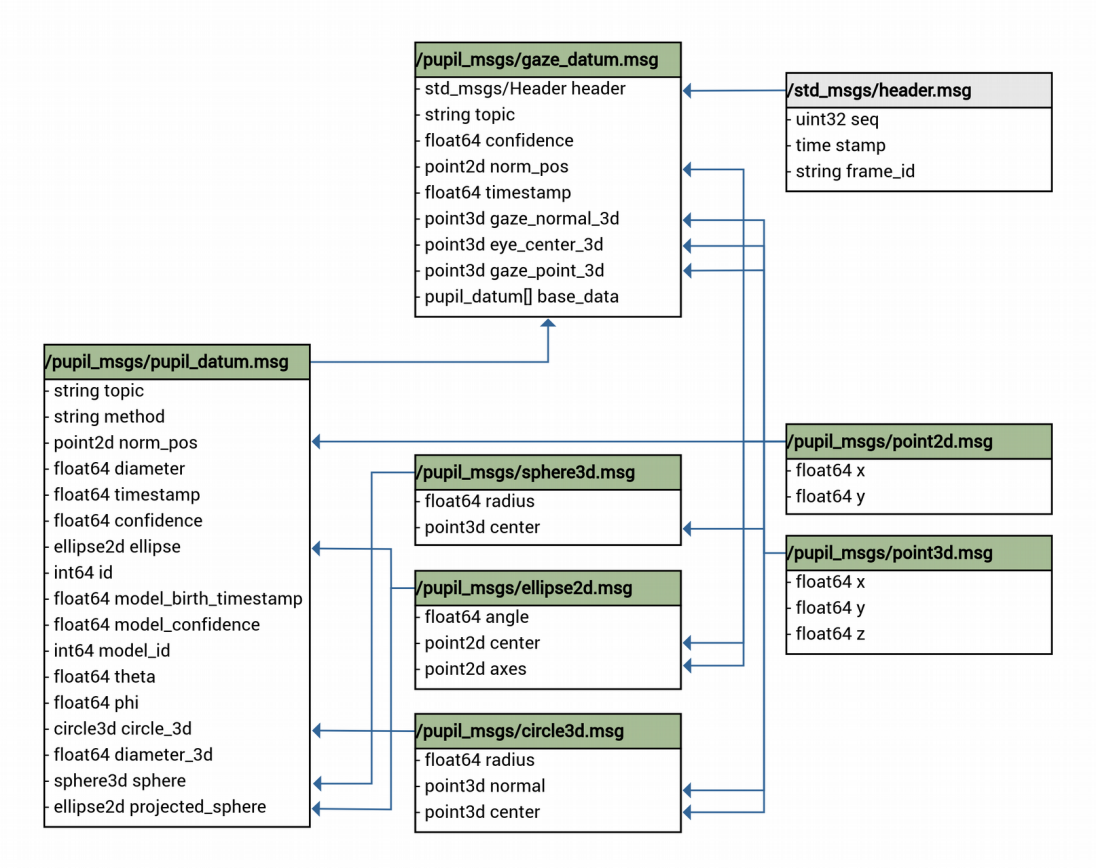


Figure 8: ROS custom message structure populated directly with each of the MsgPack Adaptors respectively designed. The generic types such as int64, float64, string and std_msg/Header are common ROS message types. Message types shaded in green represent custom ROS messages.

All of the information required is then unpacked, using the MsgPack library adaptors, directly into the respective messages counterparts, which upon conclusion are published into the ROS server.

As mentioned before, the necessary messages for the ROS environment also include the three different camera images. These images are kept as payloads in the Pupil Software camera frame message types which have the following structure.

- **Camera Frame**, message with all the information related to the cameras presented in the Pupil, one camera frame message for each individual camera;
  - **topic** - name of the topic subscribed to have access to this message;
  - **index** - incremental index sequence of the image, used for synchronization purposes;
  - **timestamp** - time of the source image, used for synchronization purposes;

- **format** - image format used, in this dissertation will always be BRG8, without compression;
- **width** - width of the image in pixels;
- **height** - height of the image in pixels;
- **raw_data** - actual serialized image data.

Since the ROS already has a message type with specialized transport for images, then instead of creating a new message type as a counterpart to the frame messages, as done for the pupil/gaze messages, this message is divided into three components, thus following the ROS design guidelines. Which for image processing involves the use of nodelets instead of normal nodes and the specialized image transport strategies which include the compressed and theora transports. This specialized strategies require the use of the ROS messages types *sensor_msgs/Image* and *sensor_msgs/CameraInfo* which already include most of the internal Pupil Software frame message data. Although not really necessary a ROS message type is also created to store all the data of the original frame message, with the exception of the actual image data, in order to preserve a similar message structure as observed in Pupil Software. Which results in the following group of ROS messages used for each of the three Pupil cameras.

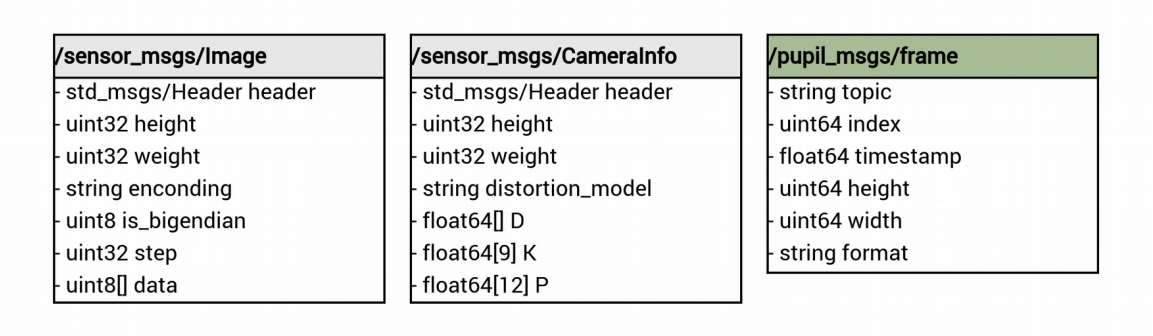


Figure 9: Messages used for each of the Pupil cameras. Message types shaded in green represent custom ROS messages.

Now, it is also important to verify the capability of the communication module created. In order to do so a latency test is executed individually for each of the camera frame messages, important to note that the *gaze_datum* message latency is considered negligible since, taking in consideration that its message size is around 600 Bytes while the eye *frame* message has 480 KBytes and their published frequency is the same, the estimated latency for the *gaze_datum* should be close to x800 lower than the eye *frame*

message. Furthermore, the latency considered also takes into account the time spent parsing the received messages and the republishing into the ROS server.

Table 3: Communication and processing latency while using the ROS Pupil Package.

	Eye Frame	**World Frame**	**Depth Frame**
Average (ms)	3.58	16.01	3.77
STD (ms)	2.54	6.5	1.71
Expected FPS	120	30	30

It can be observed, that communication between the Pupil Software and the ROS framework is very time consuming which means that frames are dropped and the resulting processing FPS are be unstable, as such, a straightforward modification to the Pupil software is necessary to limit the transmission rate, this limit is only desired in the world RGB and Depth streams.

4.2 Pose Estimation using the RGB-D camera

Taking into account the previous section, then all the camera *frame* and *gaze_datum* messages are now accessible in the ROS. So considering that this dissertation aim is to control a robotic arm by inferring from this data, then its necessary to transform this data, which is tied into the Pupil headset workspace, into the same workspace, which in this case is the coordinate system of the base of the robot used. However, this transformation is problematic as it requires knowledge of the Pupil whereabouts in the robot coordinate system, and since the Pupil is the coupled to the user itself which is mobile, then this estimation needs to be computed in real-time by correlating the static world, in the robot base coordinate system, which is known in advance, with the world as seen from the Pupil world RGB and Depth cameras.

This correlation is usually done with a Perspective-N-Point (PNP) algorithm. This estimates the pose of the camera used by correlating 3D points, at known locations in the world, with their corresponding 2D points observed in the RGB image. Considering the knowledge of the intrinsic parameters of the camera used it is possible to achieve a 3D pose estimation with at least 4 pairs of corresponding points between the 3D and 2D workspaces. However, this estimation is highly dependent on the quality of both the camera calibration

and of the 2D point detection. As such, since the Pupil world camera RGB stream is not calibrated in the Pupil software and the gaze calibration methods are applied directly on the distorted image, plus taking in consideration that the camera has a rolling shutter and is positioned in a mobile structure, then this pose estimation method is not the most recommended option. Especially since using it completely discards the World Depth data.

Since the World Depth information is provided, then this pose estimation can be simplified to the estimation of a Rigid Body transform. Which in practice means finding the most suitable translation and rotation for fitting two equal sets of 3D points, in this case the same world points as seen from different coordinate systems. Then, with that end in mind, its necessary to have pairs of 3D corresponding points between the world coordinate system and the points generated by the Depth image in the Pupil coordinate system. To achieve these matching points the Aruco library from OpenCV is used, in which is implemented a detection algorithm for custom 2D markers, similar to 2D QR codes, with individual IDs. Using this library detection algorithm directly in the distorted world RGB camera image provides the pixels coordinates for each of markers corners with its respective IDs, only these points are undistorted. Now, considering the Depth image information has been aligned into the RGB image plane using the known intrinsic and extrinsic parameters for the two cameras, then a Depth value will exist for each of the previously obtained 2D markers pixel coordinates, resulting in a 3D point for each of these in the Pupil coordinate system. Now, with the corresponding pair of points defined, a straightforward implementation of the chosen Rigid Body transformation algorithm will provide the pose estimation for the Pupil in the World coordinate system.

Considering the dependence of the algorithm in the quality of the depth information then the observed offset in the figure (5) should be corrected. Thus a polynomial is fitted into the observed depth to correct the previous depth error, resulting in the new depth value $\hat{z}$.

$$\hat{z} = -0.0158z^3 + 0.0798z^2 + 0.8361z + 0.0933 \qquad \text{(Equation 4.1)}$$

This results in the correction of the depth information seen in figure (10) to,

Table 4: Corrected depth values. Average from 100 frames, in millimetres.

	750mm	**1250mm**	**1750mm**	**2250mm**	**2750mm**
Average (mm)	749.9	1250.1	1749.2	2249.3	2748.4
STD (mm)	1.9	2.7	6.8	11.0	26.3
Average Error (mm)	-0.1	0.1	-0.2	-0.7	-1.6

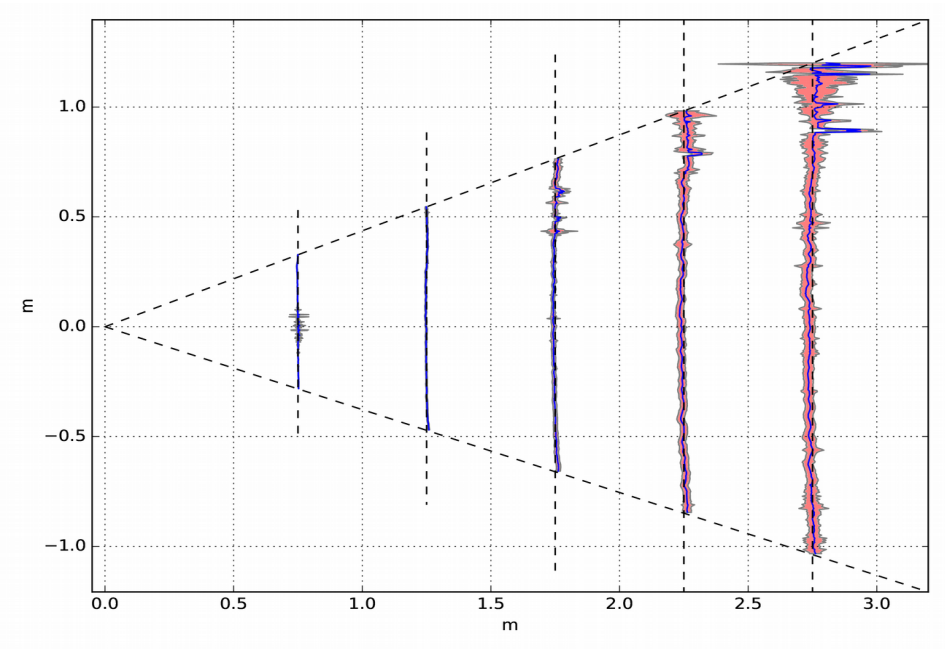


Figure 10: Projection of the corrected depth values using the specified polynomial.

Note that the depth also has a slight radial distortion, however the correction for such distortion would require a higher complexity polynomial, since the previous 1D fitted polynomial, $p(z)$, would have to be multivariate, $p(u, v, z)$, for which a more detailed data collection would be required.

As mentioned, the matching points for the Rigid Body transformation are obtained by employing a detection algorithm implemented in the OpenCV Aruco Library, which makes use of binary square fiducial markers that can be printed and placed in the world. Considering the FOV of the World camera and its advised working distance, maximum of 3.5m indoor, then markers with 5.6cm sides are adequate. Using the robot base as the world origin, then the markers arrangement chosen, for a 160x80cm table, is a grid pattern of 35x30cm, as represented by the figure (11).

Three additional markers are also placed in the robot base corresponding to the positions, ***#14*** at (0.043, 0, 0.15), ***#15*** at (0, -0.043, 0.15) and ***#16*** at (-0.043, 0, 0.15).

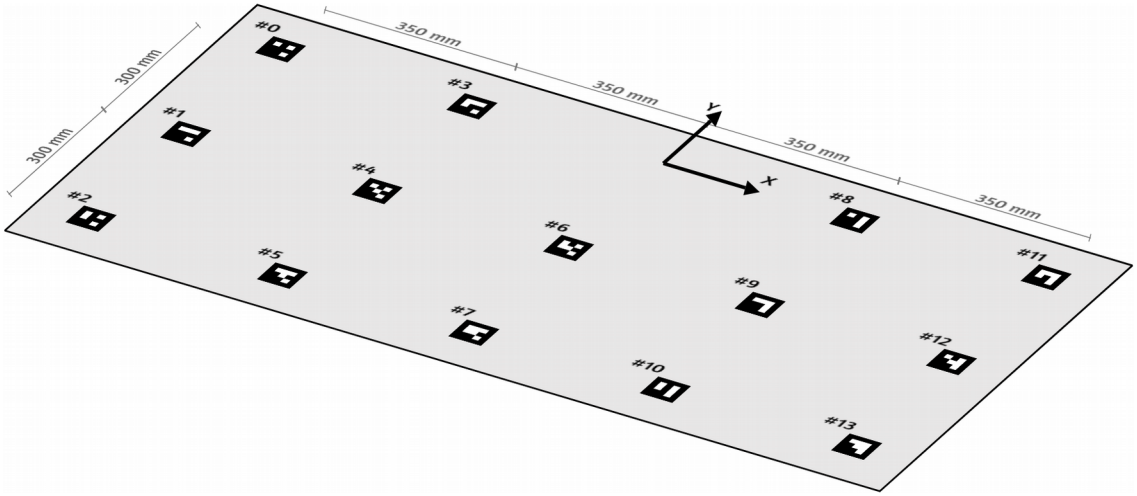


Figure 11: World Aruco markers grid placement.

Now, with the depth values corrected, and the static world markers location defined the previously outlined process results in the following pseudo-code:

```
Pseudo-Code for pose_estimation()

#[Wx,Wy,Wz,id] is the known World location of each Marker
#Extract the [x,y,z,id] from the RGB and Depth frames
[u_d,v_d,id] = aruco_detect_markers( rgb_image )
[u,v] = undistort_rgb_pixels( [u_d,v_d] , rgb_parameters )
[depth] = align_depth_into_rgb( depth_image, rgb_parameters, depth_parameters )
For Each in [u,v,id]:
    [Cx,Cy,Cz,Cid] = project_rgb_points_with_depth( [u,v], depth,
rgb_parameters)

#Estimation of the Camera Pose
[rotation, translation] = rigid_body_transformation( [Cx,Cy,Cz,Cid],
[Wx,Wy,Wz,Wid] )
```

```
Pseudo-Code for rigid_body_transformation() using Horn Algorithm

W_Centroid = calculate_world_centroid( [Wx,Wy,Wz,Wid] )
C_Centroid = calculate_camera_centroid( [Cx,Cy,Cz,Cid] )

W_Vectors = world_vectors_from_centroid_to_markers( [Wx,Wy,Wz,Wid], W_Centroid )
C_Vectors = camera_vectors_from_centroid_to_markers( [Cx,Cy,Cz,Cid], C_Centroid )

C = calculate_covariance( W_Vectors, C_Vectors )

#Construct N as per the Horn Algorithm [4x4 Matrix]
N = [C(1,1)+C(2,2)+C(3,3),  C(2,3)-C(3,2),  C(3,1)-C(1,3),  C(1,2)-C(2,1);
     C(2,3)-C(3,2),  C(1,1)-C(2,2)-C(3,3),  C(1,2)+C(2,1),  C(3,1)+C(1,3);
     C(3,1)-C(1,3),  C(1,2)+C(2,1),  -C(1,1)+C(2,2)-C(3,3),  C(2,3)+C(3,2);
     C(1,2)-C(2,1),  C(3,1)+C(1,3),  C(2,3)+C(3,2),  -C(1,1)-C(2,2)+C(3,3)];

#Rotation is the quaternion from the N matrix Eigenvector with greater Eigenvalue
[EigenVectors, EigenValues] = eig( C )
rotation = Quaternion( EigenVectors(max(EigenValues)) )
translation = W_centroid + rotate( inverse(rotation), C_Centroid )
```

Since ROS internally uses Quaternions, then the algorithm presented in [23], which estimates the rotation directly in quaternions, becomes the obvious choice for the rigid body transformation estimation, named Horn algorithm in subsequent parts of this dissertation.

To serve as comparison, with the Pupil stationary, 100 iterations of the aforementioned Horn algorithm and the PNP algorithm implemented in OpenCV are computed. With these it is possible to obtain a relative comparison between the two using the resulting standard deviation. For completion purpose, the 100 iterations are executed with different number of observable markers, **4 Markers** (*#3, #5, #8, #10),* **10 Markers** (*#1, #3, #5, #6, #8, #10, #12, #14, #15, #16),* and **17 Markers** (*all markers).*

Table 5: Relative comparison between the results obtained using the PNP and Horn algorithms.

Nr of Markers	**4**	**10**	**17**
Horn (RGB-D)			
Translation (mm)	(1.2, -0.776, 0.74)	(1.177, -0.759, 0.758)	(1.18, -0.762, 0.76)
STD (mm)	(9.2, 9.05, 7.53)	(7.07, 7.56, 4.35)	(4.3, 4.68, 2.47)
Rotation (Uq)	(-0.318, 0.083, 0.908, 0.259)	(-0.327, 0.08, 0.907, 0.254)	(-0.327, 0.084, 0.906, 0.256)
STD (Uq x0.001)	(3.29, 1.92, 1.47, 2.67)	(2.23, 3.23, 0.86, 2.27)	(1.16, 1.07, 0.57,1.69)
Time Spent (us)	26.53 µs	32 µs	29.04 µs
PNP (RGB)			
Translation (mm)	(1.066, -0.7, 0.689)	(1.174, -0.773, 0.739)	(1.171, -0.771, 0.732)
STD (mm)	(320, 210, 210)	(5.03, 4.93, 6.62)	(1.15, 1.46, 1.79)
Rotation (Uq)	(-0.301, 0.078, 0.83, 0.321)	(-0.322, 0.076, 0.907, 0.259)	(-0.321, 0.078, 0.907, 0.260)
STD (Uq x0.001)	(90, 20, 250, 200)	(2.25, 1.41, 1.12, 2.33)	(0.737, 0.53, 0.37, 0.67)
Time Spent (us)	162.26 µs	367.68 µs	437.59 µs

From the results it is possible to conclude that both algorithms are very precise, with the predominant difference being the computation time and the lower precision of the PNP algorithm for low number of markers. It must be noted that, although the computation time displayed serves as a comparison between the algorithms, the time spent detecting the markers using the Aruco Library is not included, given that both algorithms use it. This is also the heavier operation which greatly affects the final results since it takes on average 0.076 seconds to compute a single frame, including the previously noted communication and processing latency then the maximum pose estimation becomes less than 3 FPS.

Must be noted that although the previous table serves has a comparison between the two algorithms, the accuracy cannot be compared since a baseline does not exist.

4.3 Blink and Fixation Detection

Taking into consideration that all the necessary data created by the Pupil Software has been extracted and transformed into the respective custom message types and published into the ROS server, then it is finally possible to use the ROS framework to create modular programs to process the aforementioned data. For which is important to first define the specific eye movements that the system should recognize as commands and actuate. Since the perceptible types of movements the eye can make are rather limited, and that it is undesirable to have the robot end-effector tracking the gaze pose, then the eye blinks is the simplest method to use as a control system. To accomplish this a ROS nodelet named ***blink_detection*** is designed with the objective of preprocessing the *gaze_datum* message data, placing the following results into a new custom ROS message named *eye_status*.

First it is necessary to define the various desired eye states to detect for which to populate the new ROS message (12). This is constructed with copies of the ***confidence***, ***gaze_normal_3d*** and ***eye_center_3d*** found in *gaze_datum* message plus the newly computed parameters.

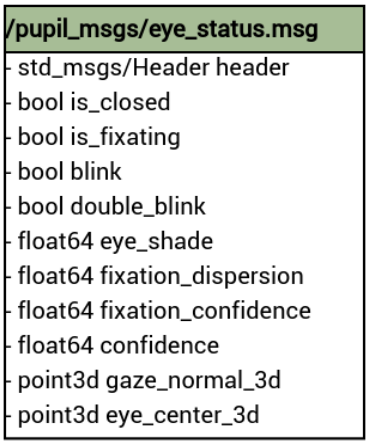


Figure 12: Eye status custom ROS message diagram.

Note that although the confidence value obtained directly from the Pupil Software could also be used to estimate the eye closure and blink detection, it is an artificial confidence value which is highly noisy and depends on the quality of the Pupil Software calibration which ends up resulting in lower quality of blink detection. This can easily be seen in the figure (13), where the confidence values provided by the Pupil Software are

presented. A caveat must be made in relation to the confidence values observed in (13) which are the result of an excellent calibration and can be considered unusually stable, usual values tend to have higher noise.

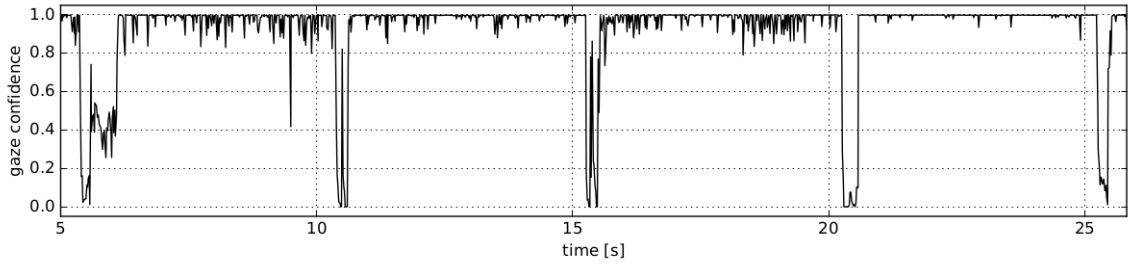


Figure 13: Example of the confidence values obtained directly from the Pupil Software for the previous double blink experiment.

Still, the observed confidence values serve to show the inconsistency of the confidence values from the Pupil Software. Considering that the double blink detection is crucial, then it is not possible to use the confidence values observed, as can be seen from the lack of detail regarding the double blink occurring at 20 seconds, then a new blink detection system is necessary.

As such, the parameters that compose the custom ROS message (12), are obtained as follows:

The ***is_closed*** flag, represents the closure of the eye, is calculated using the average of the grayscale values inside a circular window of size ***shade_window_size*** pixels around the detected pupil center, this window is displayed by a red circle in the blink demo figure (14). The resulting grayscale average is normalized, $x[k]$, and then filtered using a low pass filter (4.2), where the α is the parameters named ***shade_lowpass_alpha.*** Finally the filtered values, $\bar{x}[k]$, are validated using a threshold, parameters named ***shade_threshold,*** since the pupil is darker than the rest of the observed features when the eye is closed, which results in the figures (15) (16), where the detected average grayscale are represented in junction with the locations where the threshold is exceeded.

$$\bar{x}\,[k] = (1-\alpha)\,x\,[k] + \alpha\bar{x}\,[k-1] \qquad \text{(Equation 4.2)}$$

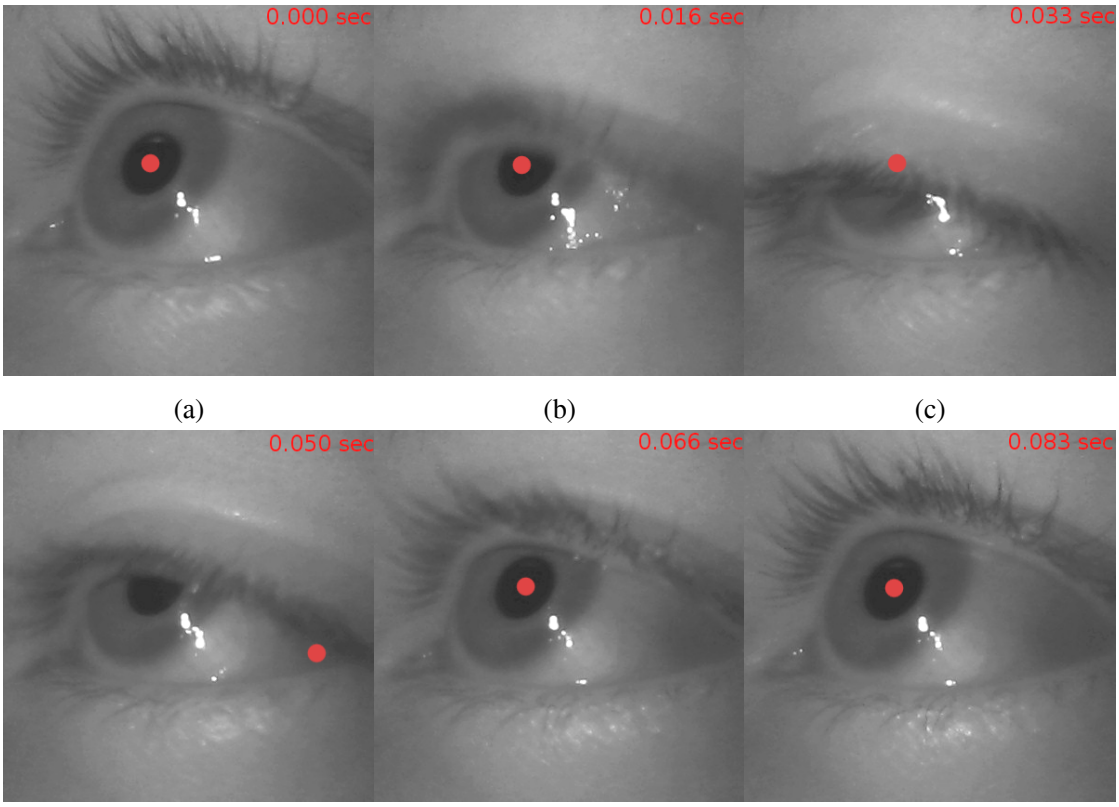


Figure 14: Six frames of the eye camera at 60 FPS capturing an eye blink and the correspondent pupil detected center and the considered circle for eye_shade computation.

The ***blink*** flag, represents a valid eye closure duration. As such, if the duration of the eye closure obtained using the ***is_closed*** flag is within the parameters ***minimum_blink_duration*** and ***maximum_blink_duration*** then the blink is triggered once.

The ***double_blink*** flag, just like the ***blink*** flag is triggered if two sequential blinks are detected and the duration between these falls within a predefined acceptable range, which is defined by the blink detector parameters ***minimum_double_blink_wait*** and ***maximum_double_blink_wait***.

To test the accuracy of both the single and double blink detection, an experiment where the user is asked to change focus and blink once when a 5 second period sound signal occurs is implemented, this is repeated with double blinks. The resulting data can then be used to improve the detection parameters until the detection results are considered acceptable.

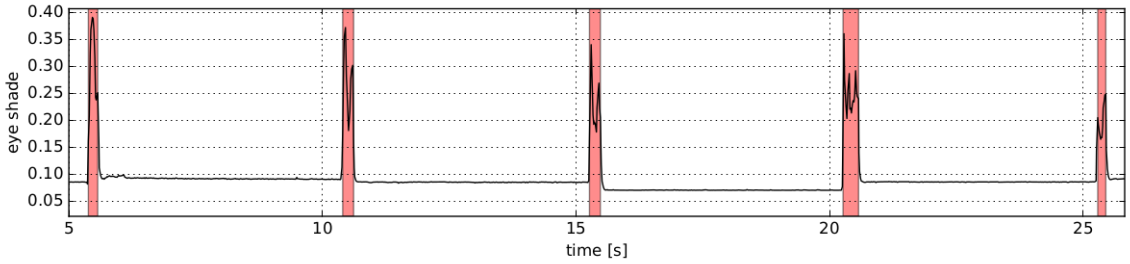


Figure 15: Normalized average grayscale detected in a circular window with radius N pixels around the detected pupil center during the single blink experiment. The locations where the defined threshold are presented shaded in red.

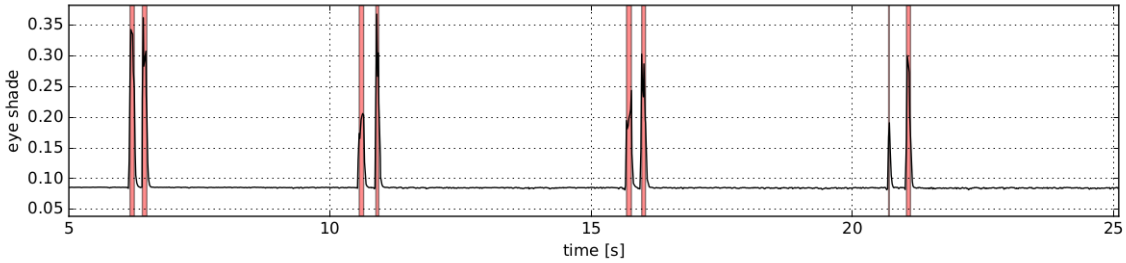


Figure 16: Normalized average grayscale detected in a circular window with radius N pixels around the pupil center during the double blink experiement. The locations where the defined threshold are presented shaded in red.

Finally the ***is_fixating*** flag uses the normalized 2D gaze point in the ***gaze_datum*** message to calculate the 2D gaze dispersion considering a temporal window of ***minimum_fixation_duration*** seconds, from which results the selection of N points in the image plane, $(u_i, v_i)_{i=1,\dots,N}$. This dispersion is defined by the norm of a two axis weighted standard deviation, weighted by the confidence from the Pupil Software gaze detection that is represented by C_i in (4.3). This standard deviation is obtained for both u and v values from the normalized gaze position and the estimated 2D dispersion corresponds to (4.4).

$$\sigma_{w,u} = \sqrt{\frac{\sum_{i=1}^{N} C_i \left(u_i - \overline{u}_i\right)^2}{\sum_{i=0}^{N} C_i}} \qquad \text{(Equation 4.3)}$$

$$dispersion = \sqrt{\sigma_{w,u}^2 + \sigma_{w,v}^2} \qquad \text{(Equation 4.4)}$$

The ***is_fixating*** flag is then obtained using a threshold on this dispersion values, parameter named ***dispersion_threshold***. Considering that in the experiment for blink detection the user is also asked to change focus then, from the previous experiment results, the fixations can also be estimated from the resulting dispersion curve, as observed in (17).

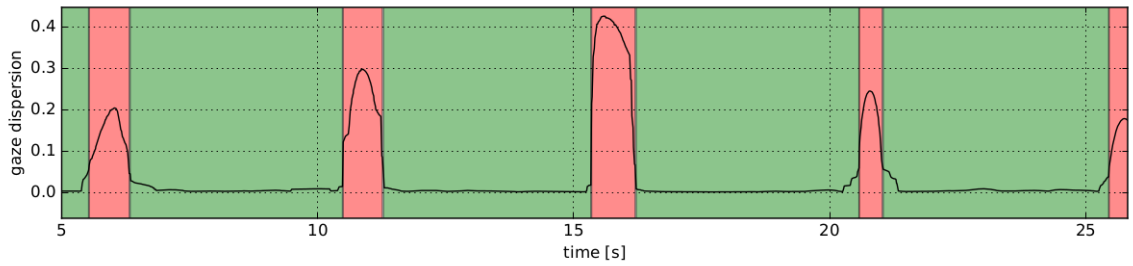


Figure 17: Calculated 2D gaze dispersion for the blink detection displayed in the figure 15. Locations shaded in red represent threshold excedance which means lost of fixation.

Important to note that this weighted standard deviation will reduce the effect the eye closure has on the dispersion calculation since the Pupil Software confidence will drastically drop.

The blink detector parameters used previous are,

Table 6: Parameters used in the blink detector module.

Parameter Name	Value
shade_window_size	10 (px)
shade_lowpass_alpha	0.4
shade_threshold	0.15
minimum_blink_duration	0.03 (s)
maximum_blink_duration	0.35 (s)
minimum_double_blink_wait	0.04 (s)
maximum_double_blink_wait	0.5 (s)
minimum_fixation_duration	1.0 (s)
dispersion_threshold	0.07

These parameters serve only as reference since these are user-specific, as such the corresponding results can not be duplicated. For each new user a new blink detection profile must be calibrated.

4.4 Gazebo Simulation

As mentioned in the section (1.3), a simulation environment is also implemented using Gazebo by mimicking the work done in the Pupil Software, section 3.3, the Pupil to ROS communication, section (4.1), and the Pose estimation, section (4.2), programs. Resulting in the following 3 major components of the simulation:

- Gazebo RGB-D camera simulation;

  Modified version of the Kinetic camera simulation plugin for gazebo, used to synthesize the depth stream, it is notable that the this visual plugin has a noise function included for which the values obtained in the section (3.2.2) are used.

- Simulation of the Pupil movement in the world using a Space Mouse;

  It is important for the simulated camera to be mobile, mimicking the mobility of the user, which can be easily achieved using a space mouse, a 6DOF joystick to directly control the velocity of the simulated camera pose.

- Simulation of the Eye movement, using a normal mouse;

  The most important component of this simulation is obviously the synthesis of a fake eye gaze. In the Pupil Software the 3D gaze vector is projected into the RGB image for display purposes, however to do so it is necessary to consider the intersection of such gaze vector with an artificial orthogonal plane, in this case the RGB image plane. However, for correct function of such transformation it is necessary that the origin of the gaze vector and the considered plane coordinate system is the same, which means that the eye center is placed in the origin of the camera coordinate system. From this results a gaze vector that is simply the line projection of the image pixel were the computer cursor is placed. Furthermore, pressing the mouse button is configured to mimic the eye closure by modifying the necessary confidence parameters and boolean flags in the synthetic ***eye_status*** message.

Considering a similar workspace arrangement as defined in section (4.2), and that the robot is also spawn into the simulation, with either a jointspace torque or jointspace position controller, then the simulation interface becomes a 2D image from which the gaze line can be controlled using the computer cursor, and a 3D view, using Rviz, of the generated pointcloud and projected 3D gaze.

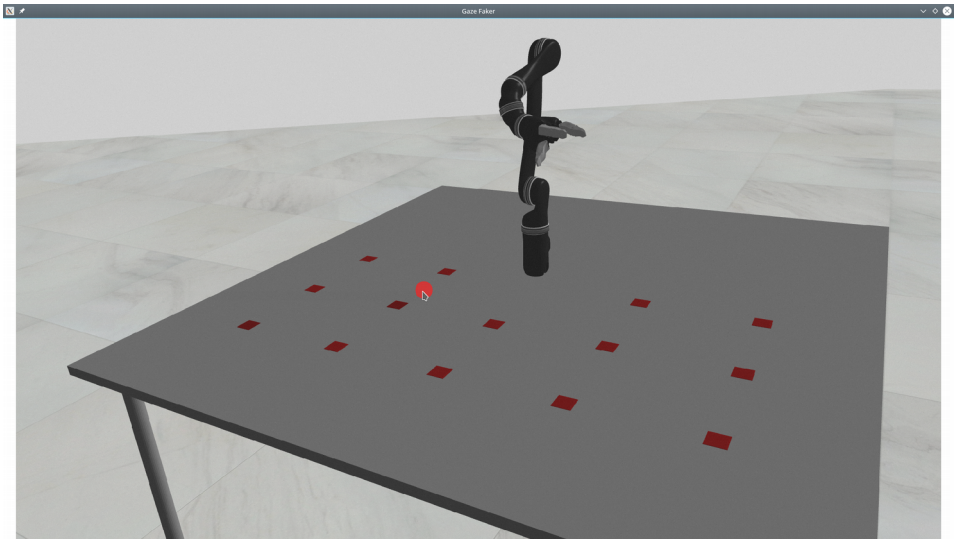


Figure 18: RGB image generated from the gazebo plugins, where the cursor position (circled in red) is used to simulate the eye focus point.

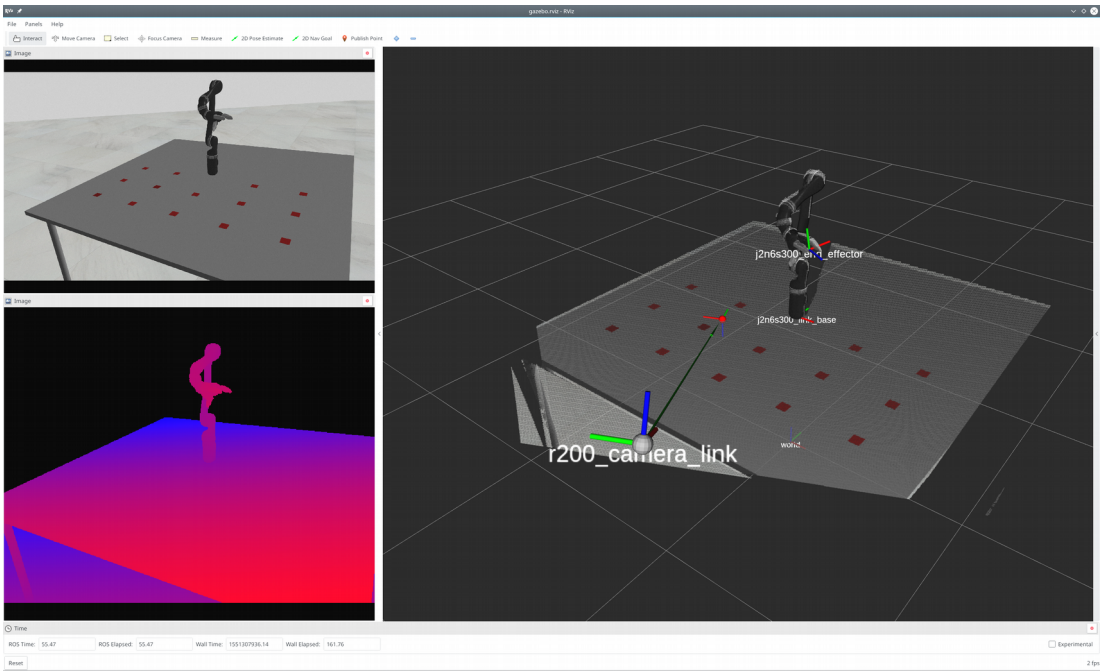


Figure 19: Corresponding views using Rviz. In the Top-Left view the RGB image published into the ROS. In the Bottom-Left view the depth image published into the ROS. In the Right view the generated pointcloud, where the 3D gaze is visualized by the green line intersection the pointcloud as chosen by the cursor in (18).

5 ROS Gaze Package

Considering the eye information as preprocessed both in the simulation and real environments then it is now necessary to start the processing of such data into the requirements for robot control, which means the requires the complete definition of the chosen control method and the robot in which these are applied.

The robot used is the Jaco^2 developed by Kinova Robotics [24], a company which focus on assistive/rehabilitation devices. This robot has 6 DOF with a 3 finger removable end-effector. Its most notable characteristics are the unlimited actuators rotation, its lightweight construction, less than 5 Kg, its low power consumption, less than 25W average and its exceptional low footprint. This robot also has several controllers implemented internally as seen in (21.b), these are hard-coded into the robot firmware and can be adapted using specific API functions, which remove the necessity of a real-time enabled computer for correct robot control. As such the only requirements, considering the cartesian position controller as the desired control method, are the desired end-effector pose and a duration for the robot to complete the respective movement.

As an alternative to the internal trajectory controllers, it is also viable to use a custom controller by using the direct torque control mode, however while in the torque control mode, the Kinova Joystick (20) is disabled, which is not desired thus the previous mentioned selection.

It is worth noting that the aforementioned joystick only has 3 degrees of control, requiring the switching between modes to achieve control of the complete 6 DOF of the robot plus an additional mode for control of the gripper. This results in 3 separate modes, position, orientation and finger modes, and considering the joystick assembly (20), none of the above modes are fully intuitive. As an example, in the position mode, the robot X-Y is controlled using the respective linear joystick displacements while the its Z axis is controlled using the rotation of the joystick.

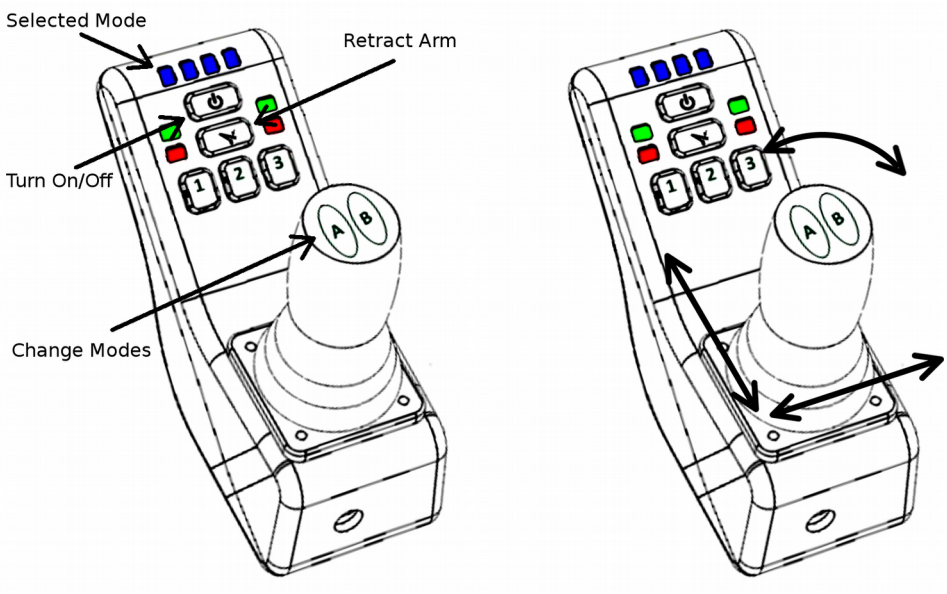


Figure 20: Kinova Joystick description and control movements. (a) Description of the most important buttons. (b) Example of the control movements of the Kinova Joystick.

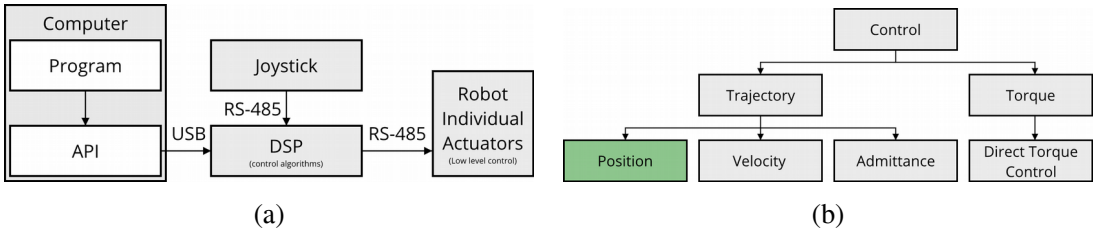


Figure 21: Control diagrams in reference to the Jaco Robot. (a) Computer to robot communication structure. (b) Types of internal controllers hardcoded into the robot firmware, all types have both angular and cartesian versions.

Further information about the Kinova Jaco[2] robot and its specification can be seen at [D].

As mentioned before the internal controller chosen for this task, shaded in green, is the cartesian trajectory controller, which simplifies the requirements to a cartesian end-effector pose and a span of time for the robot to complete the motion.

5.1 Gaze Intersection Pose

Considering the aforementioned requirements then it is necessary to extract a 3D pose from the gaze information obtained from the Pupil Software. Which can, if the ***blink_detector*** recognizes the correct commands, be used to operate the robot. As described previously, the data extracted from the Pupil Software has already been pre-processed from which results the ROS message names ***eye_status*** where all the necessary variables have been compacted including the 3D gaze vector, $\vec{g}$, that represents the gaze line that passes through the 3D eye center, also present in the message.

Considering that the desired output is the intersection pose between the gaze line and the point cloud then, considering the depth noise which has been observed in the section (3.2.2), it is necessary to consider more than one point when resolving this intersection, which means the definition of a maximum acceptable minimum distance from each considered point to the gazeline for which it is considered part of the intersection. This is analogous to a cylinder of acceptable points with the gaze line as its center axis and radius equal to the aforementioned maximum distance. To accomplish this, it is necessary to compute the minimum distance from every point in the pointcloud to the gaze line, which, considering the gaze line vector as $\vec{g}$ and that $\vec{p}_i$ represents the vector constructed from the eye center to the i point in the pointcloud, can be resolved by solving the following equation.

$$d_i = \frac{|\vec{p}_i \times \vec{g}|}{|\vec{g}|} \qquad \text{(Equation 5.1)}$$

Now, considering as observed points only those for which the distance d_i is less than the predefined value, then using this new list of valid points it is possible to estimate the 3D intersection point, between the gaze and the point cloud, as its positional average and the respective orientation by analysing its local features.

Since the desired pose Z axis should be orthogonal to the observed plane and off-setted by a predefined distance. The observed plane, equivalent to the best fitted plane to the observed points, can be estimated using the Principal Component Analysis (PCA) algorithm from which results 3 vectors, which define the orientation of the observed point distribution axes, and respective vector magnitudes, these are the resulting eigenvectors and eigenvalues from the covariance matrix of the previous points.

This results in the following pseudo-code:

```
Pseudo-Code for estimate_plane_normal()

[valid_points] = check_distance_to_gazeline( [pointcloud], gazeline,
valid_distance)

centroid = calculate_centroid( [valid_points] )
covariance = calculate_covariance( centroid, [valid_points] )

# Estimated plane normal is the Eigenvector of the smallest Eigenvalue
[EigenVectors, EigenValues] = eig( covariance )

plane_normal = EigenVectors[min(EigenValues)]
```

Considering a planar distribution then the points will have two axes with similar magnitudes, eigenvalues, and axes with smaller magnitude which represents the observed plane estimated normal vector. However, a single plane normal can not be resolved into a singular pose since not only does the plane normal $\hat{n}$ and $-\hat{n}$ represent the same plane but there is also a degree of freedom represented by the $\hat{n}$ vector own rotation.

To resolve this, first it is important to note that the 3D points considered for the PCA algorithm are represented in the camera coordinate system which has its Z axis, camera optical axis, orthogonal to the image plane, as such the estimated plane normal is relative to it. From this it is intuitive to select the plane normal that is opposed to the camera optical axis, the camera can only see what is facing the user, which is equivalent to choose from, $\hat{n}$ and $-\hat{n}$, the normal that has negative z component.

As such considering the rotation matrix that defines the gaze intersection pose with the point cloud in the camera coordinate system as,

$$ {}^{c}_{g}R = \begin{bmatrix} {}^{c}_{g}\hat{x} & {}^{c}_{g}\hat{y} & {}^{c}_{g}\hat{z} \end{bmatrix} \qquad \text{(Equation 5.2)} $$

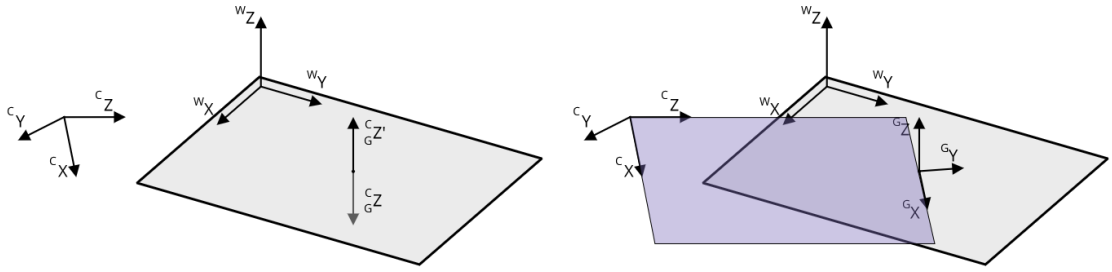
Then, the pose with the Z axis normal to the observed plane can be represented by defining the rotation matrix third column as the aforementioned plane normal.

$$ {}^{c}_{g}\hat{z} = \hat{n} \qquad \text{(Equation 5.3)} $$

Now, to fill the rest of the rotation matrix it is necessary to eliminate the aforementioned degree of freedom, which can be done by forcing the projection of the intersection pose X axis into the X-Y base coordinate system plane to coincide with its X

axis. This can be achieved using the rotation matrix orthogonality characteristics resulting in following steps,

$$
\begin{aligned}
{}^{g}_{c}\hat{x} &= (0,1,0) \times {}^{g}_{c}\hat{z} \\
{}^{g}_{c}\hat{y} &= {}^{g}_{c}\hat{z} \times {}^{g}_{c}\hat{x}
\end{aligned}
\qquad \text{(Equation 5.4)}
$$

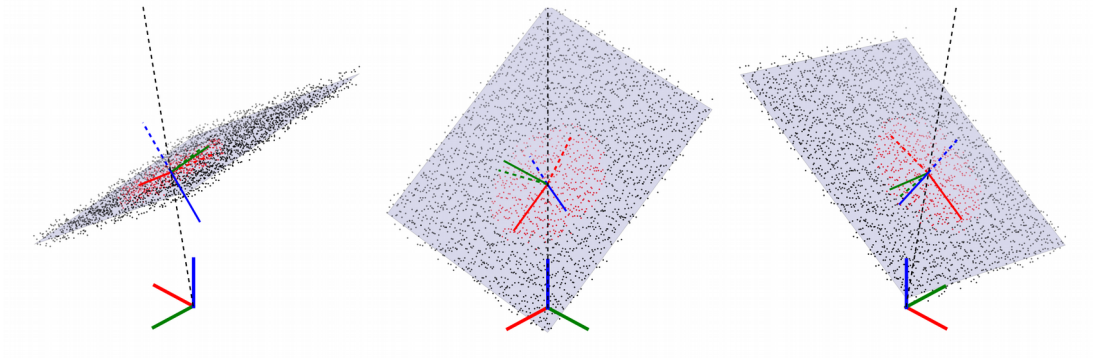


(a) (b)

Figure 22: Gaze intersection pose generation process. (a) Correcting the plane normal. (b) Gaze intersection pose definition by forcing the pose X axis projection into the X-Y base plane to be coincident to its X axis.

Which can then be grouped together according to (5.2). Finally since a similar rotation matrix ${}^{w}_{c}R$ exists that defines the camera coordinate system in the world, generated from the pose estimation algorithm in section (4.2), then the desired pose becomes fully defined, ${}^{w}_{g}R = {}^{w}_{c}R\,{}^{c}_{g}R$.

Now, considering the depth noise observed in the section (3.2.2), there is a minimum valid distance to the gaze line for which the resulting plane normal has acceptable error. To test this, the previously described procedure is simulated using the observed depth noise for a 2m distance from the camera. The intersection between a fake gaze line and this plane is then estimated according to the defined procedure.

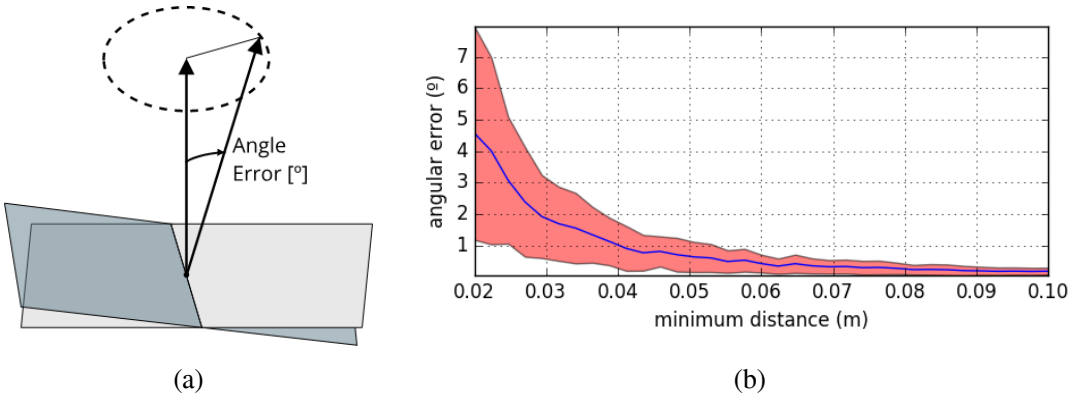


(a) (b) (c)

Figure 23: Different views of a simulated gaze intersection pose estimation. Original plane generated from the quaternion pose (0.118 - 0.299i - 0.024j – 0.946k). Dashed coordinate system represents the initial orientation extracted from the PCA, and solid line coordinate system the refined orientation.

In the figure (23), the chosen points for the PCA can be visualized in red, and the correct processing of the initial orientation estimation, according to the previously defined method, can be observed. First, in (23.a) the plane normal flipping to the desired normal opposed to the camera optical axis, dashed blue to solid blue. And in both (23.b) and (23.c) the initial intersection pose is rotated to force its X axis, dashed red to solid red, to project into the base coordinate system X axis.

From this the considered valid distance can now be modified to estimate the plane normal error, for the depth noise considered which is respective to a distance of 2m to the depth camera considering the noise observed in (3.2.2), relation to the valid distance used. Also important to define the error convention used in the figure (24) which is the angular error between the estimated plane and original plane normals. Averaging the error for each distance with the results from 100 random planes, then the resulting average and standard deviation can be used to select the distance that provides the better balance between minimum size of the objects that can be detected and the accuracy of the plane orientation estimation.



(a) (b)

Figure 24: Angle error expected between the real planes and the estimated planes normals. (a) Definition of the angle error between the original plane normal and the estimated plane normal, shaded plane. (b) Angular error observed for the depth noise at 2m in relation to the distance to the gaze line used to estimated the plane normal.

To verify the accuracy of the aforementioned process, in a real environment, a sound beep with a 5 second period is used to signal the user to change focus between a predefined sequence of known locations, 3D points and expected normal. Which in this experiment are the marked locations in the following figure (25).

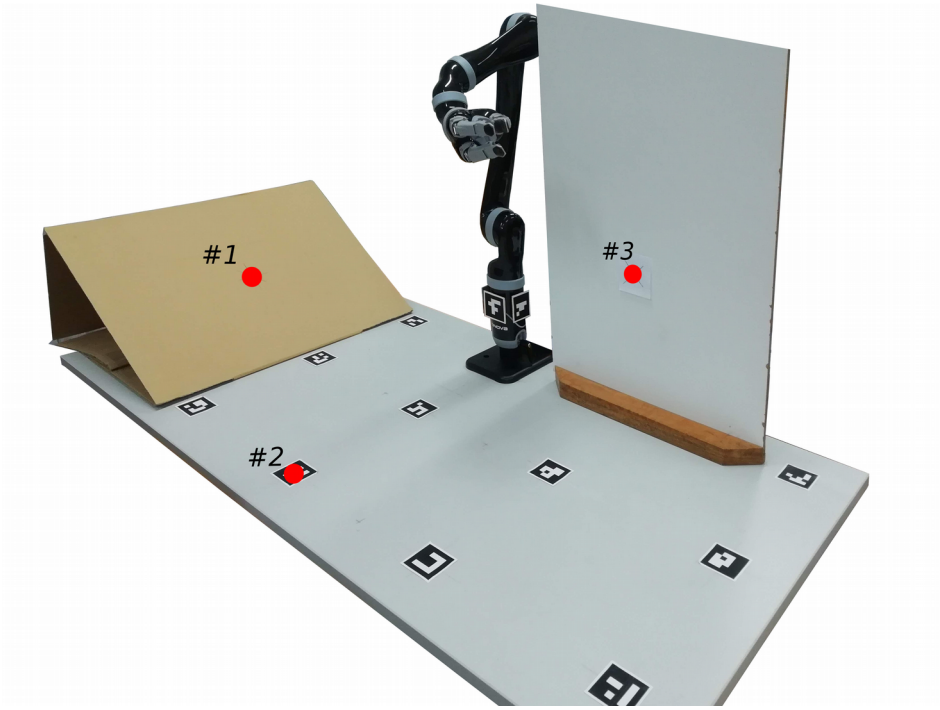


Figure 25: Photo of the arrangement for the goal pose accuracy experiment. The predefined locations and sequencing marked in red.

It is also important to note that the error present in both the table (7) and the figure (26) come as a consequence of the propagation of the gaze vector error with the errors from the eye tracker pose estimation algorithm and the innate depth noise, which also affects the pose estimation. As such it is clear that the lowest error achievable is limited by the stability of the depth values which has been previously been stated to have a standard deviation of 0.5 to 2 cm at usual working distances. Considering the angular error expected from the Pupil Software to be lower than 1º degree then the respective gaze error generated, in an orthogonal plane at 2m, adds another ~1 cm error to the gaze pose position. The results and errors presented in the following table correspond to the average obtained of the steady-state for each of the predefined locations.

Table 7: Expected, observed and error position and orientation of the estimated gaze intersection pose.

	#1	**#2**	**#3**
Position			
Expected (m)	(-0.584, -0.3, 0.144)	(0, -0.6, 0)	(0.35, 0, 0.3)
Result (m)	(-0.566, -0.346, 0.145)	(0.026, -0.609, 0.031)	(0.44, 0.023, 0.287)
Error (m)	(-0.018, 0.046, -0.001)	(-0.026, 0.009, -0.031)	(-0.094, -0.023, 0.003)
Orientation			
Expected	(0.584, 0, 0.812)	(0, 0, 1)	(0, -1, 0)
Result	(0.506, 0.019, 0.862)	(-0.018, -0.020, 0.998)	(-0.026, -0.998, -0.058)
Error	(0.088, -0.019, -0.050)	(0.018, 0.020, 0.001)	(0.026, -0.002, 0.058)

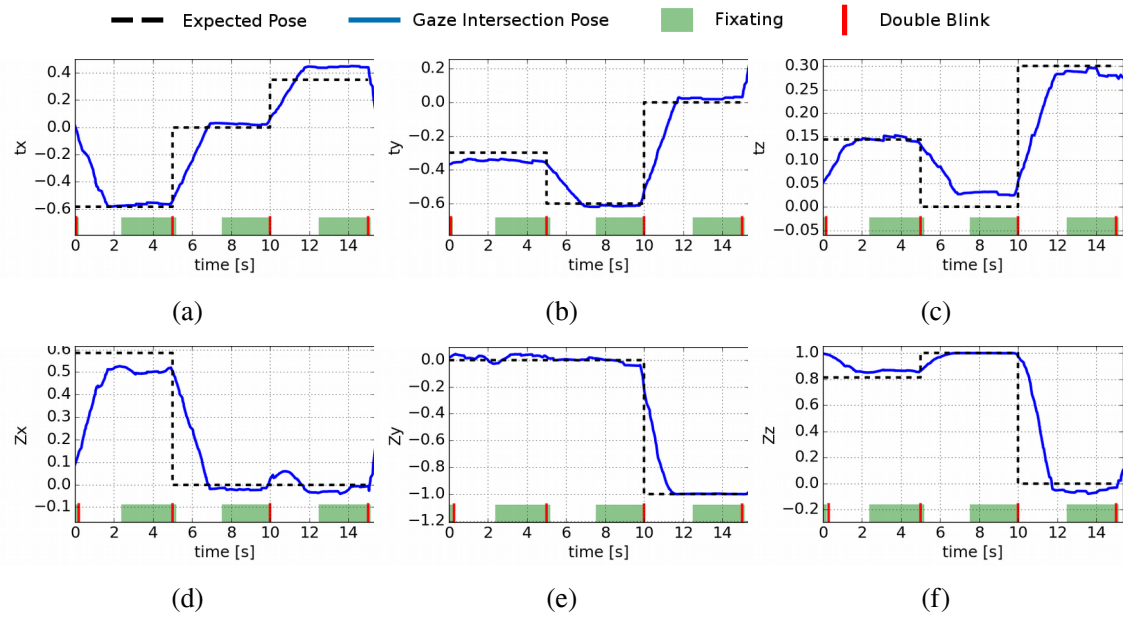


Figure 26: Gaze intersection pose experiment results for 3 predefined locations. (a)(b)(c) Position in x, y and z world coordinate system axis. (d)(e)(f) Resulting x, y and z components of the intersection pose Z axis vector. With valid fixation period signaled in green and double blink occurance time in vertical solid red line.

The previous results serve as a proof of concept for the generation of a pose, from the eye gaze and point cloud, eligible for robot control. Although errors are clearly observable in these results, considering the direct influence of both the depth noise observed in section (4.2) and the error of the eye tracker pose estimation algorithm, then the stationary error seen above can be considered acceptable. A delay is also noticeable in the generated pose, however, this is an intended result of a higher temporal period considered for the pose average, in this case the used period for pose stability was 2 seconds. Considering a lower period results in faster responses to the eye gaze however considering the low FPS of the depth frame this parameter is increased. Similarly the fixation detection also has a slight delay which also results from the temporal window considered, in this case a period of 1.0 seconds and a dispersion threshold of 0.03.

The lower FPS from the depth frame also limits the system allowing only slow head movements and even these will introduce extra errors. These are mitigated by halting any head movements before fixation and goal selection is desired.

Now considering the resulting plane normal, and the aforementioned method to create a complete pose orientation then the desired goal pose can be constructed using a simple transformation from the plane normal to the orthogonal and offseted desired pose.

5.2 Closest Marker Pose

Now, although the desired pose, orthogonal to the observed local plane with an offset, can be extracted as demonstrated previously, the quality and stability depends on the depth camera noise, the amount of points considered for the gaze intersection with the pointcloud and the accuracy of the camera pose estimation. It is also important to note that the points used for the orientation estimation must be similar to a plane which is analogous to define a limit of the curvature allowed for which a correct orthogonal pose can still be extracted correctly this means that using the previously defined algorithm for pose generation from the gaze intersection with the pointcloud will consider small objects as noise.

A resolution to this problem is achievable by replacing the previously desired pose estimation method, with a list of predefined poses from which the correct selection is attained by using the detected Aruco markers. In other words, for each single or group of aruco markers there is a predefined desired pose which can be chosen if the 3D intersection point between the gazeline and the pointcloud is inside a fixed-size bounding box around the selected marker. Important to note that the bounding box is defined in the world coordinate system orientation for static markers, and for groups of markers in their local coordinate system, groups require +3 markers as demonstrated by the box in figure (29).

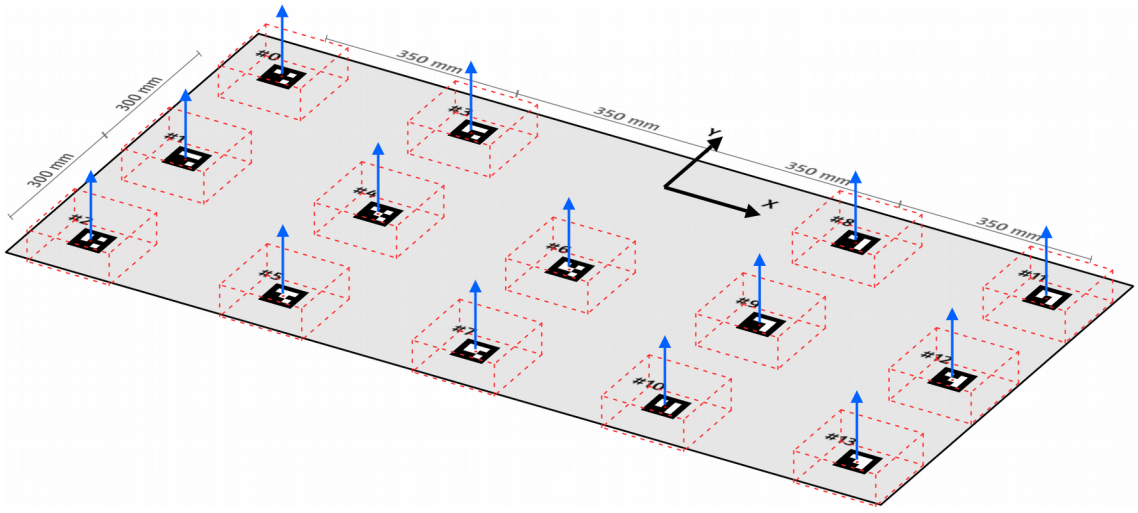


Figure 27: Static markers in the world coordinate system, with corresponding bounding boxes and respective plane normals.

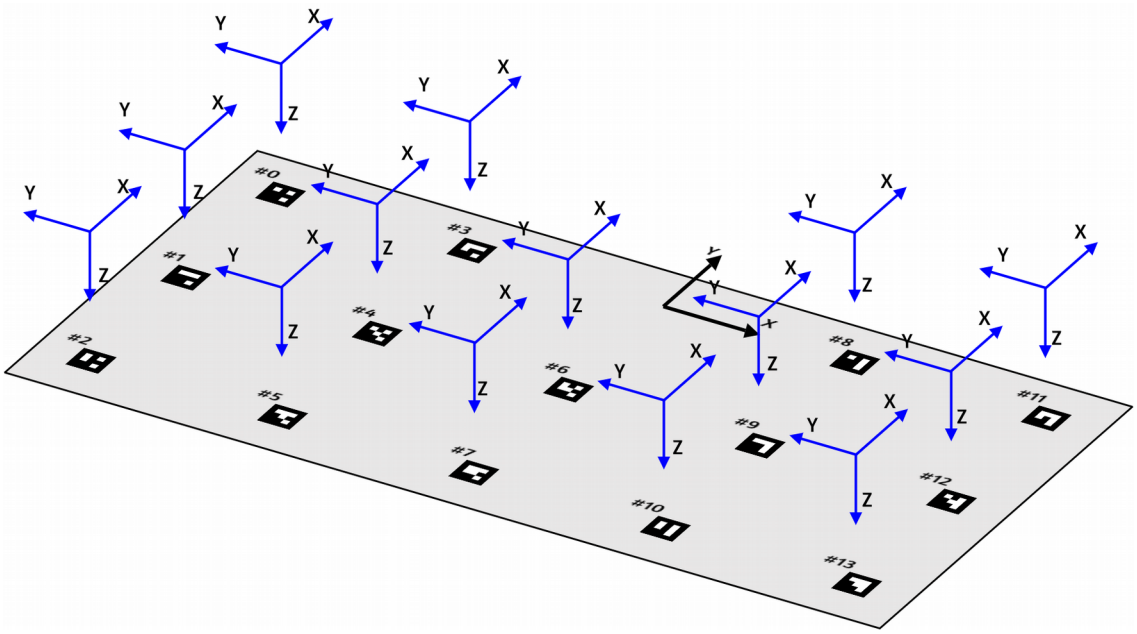
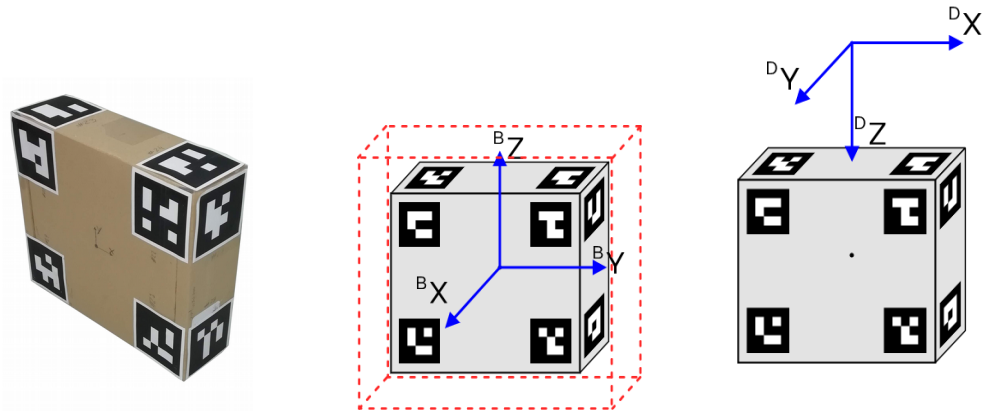


Figure 28: Static markers in the world coordinate system and respective predefined desired poses.

As mentioned an additional mobile group of markers were placed around a 22x19x6cm box to test for mobile objects. The respective Aruco markers locations are applied to a Horn algorithm, analogous to the method used in the section (4.2), to obtain the pose of the box in the camera coordinate system and subsequently the box pose in the world coordinate system.



(a) (b) (c)

Figure 29: Photo and diagrams of the box used for the closest markers method. (a) Photo of the marked box. (b) Diagram of the box, respective bounding box and local coordinate system. (c) Desired pose if box is selected.

Since the selection procedure is based on bounding boxes then the position accuracy of the gaze intersection with the point cloud observed previously in the table (7), can be used to choose the limits of the bounding box. As such a bounding box of 15x15x15cm

was chosen for the static world markers and a 32x29x16cm bounding box for the in the box group of markers, which represents a 7.5cm acceptable error for each side.

To verify the accuracy of the aforementioned method, the resulting gaze 3D point is evaluated when the user double blinks while focusing in each of the static world markers. This is repeated several times, each time with a new Pupil calibration, resulting in a heatmap which can be visualized by projecting the resulting points into the world X-Y plane which corresponds to the figure (30). The selection of the box is only displayed in the final results in the pick-and-place demo.

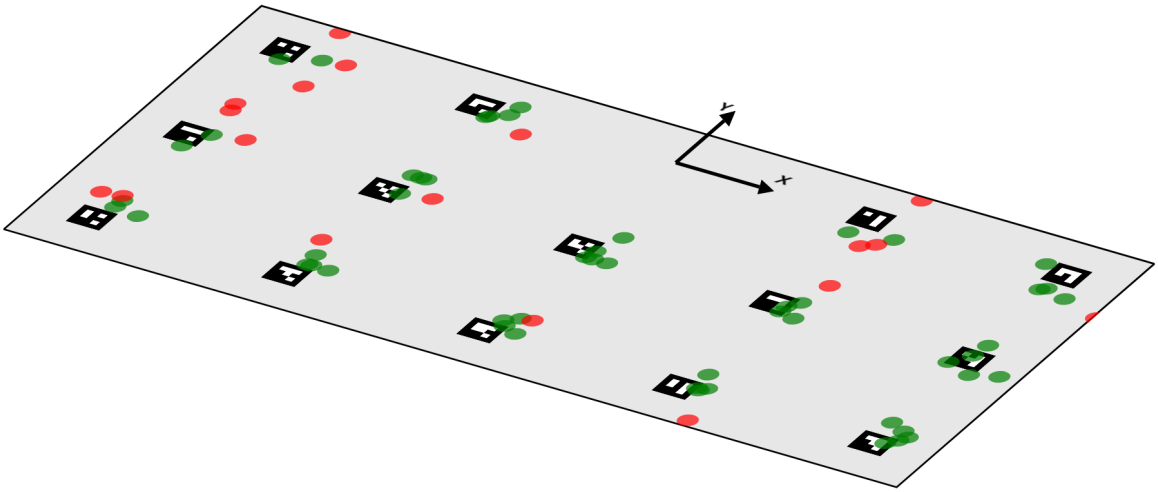


Figure 30: World static marker selection heatmap projecting the 3d gaze points into the world X-Y plane. The process is repeated 5 times for each of the world markers. Green points represent accepted selections and red represent invalid selections using a bounding box of 15x15x15cm for each marker.

The previous results serve as demonstration of the percentage of the correct selection of a marker, from this it is possible to infer the functionality of this approach which considering the correct select of the desired marker, the respective pose can then be feeded directly into the robot controller.

From the previous experiment, in which 70 selections of markers are attempted, 52 are accepted and the remaining 18 denied as these fall outside the respective bounding boxes, this results in a 74.3% selection success rate. Worth noting that the least success rate is observed in the leftmost markers, according to the figure (30). These correspond to the markers with least depth precision, according to the depth analysis in the section (3.2.2), since they occur in the left side of the depth image.

6 Results

Considering the complete system as defined in the figure (1), then with the individual modules as defined in the previous chapter the final results and robot control can finally be executed. Considering the extensive testing of each of the components, this chapter serves only as a demonstration of the complete system in operation.

Considering the different options for generating a desired pose then this chapter is also divided accordingly resulting in 2 different applications, gaze extraction directly from the pointcloud and pose generation from the static markers and the mobile box, which is used in a pick-an-place demo.

6.1 Gaze Intersection Pose

Using a similar workspace arrangement as seen in (25), with the vertical wall moved to accommodate for the robot movements. Then the expected results, since the robot control is external and assumed to have limited error, will be consistent with the former results (26) with the main difference being a transformation to the observed plane pose to obtain the previously mentioned robot goal which is desired to have its Z axis vector normal to the plane and be offsetted by 15cm. This pose can then be directly feeded into the internal robot control, when the user signals as such.

Identical to the experiment aforementioned, a 5 seconds period sound signals the user to change focus according to predefined locations. Using the following image (31) as reference, the sequence adopted follows the marked locations as ***#1*** -> ***#2*** -> ***#3***.

- Location ***#1*** with position (-0.496, -0.3, 0.266) and orientation (-0.584, 0, -0.812).
- Location ***#2*** with position (0, -0.6, 0) and orientation (0, 0, -1).
- Location ***#3*** with position (0.70, -0.3, 0.3) and orientation (-1, 0, 0).

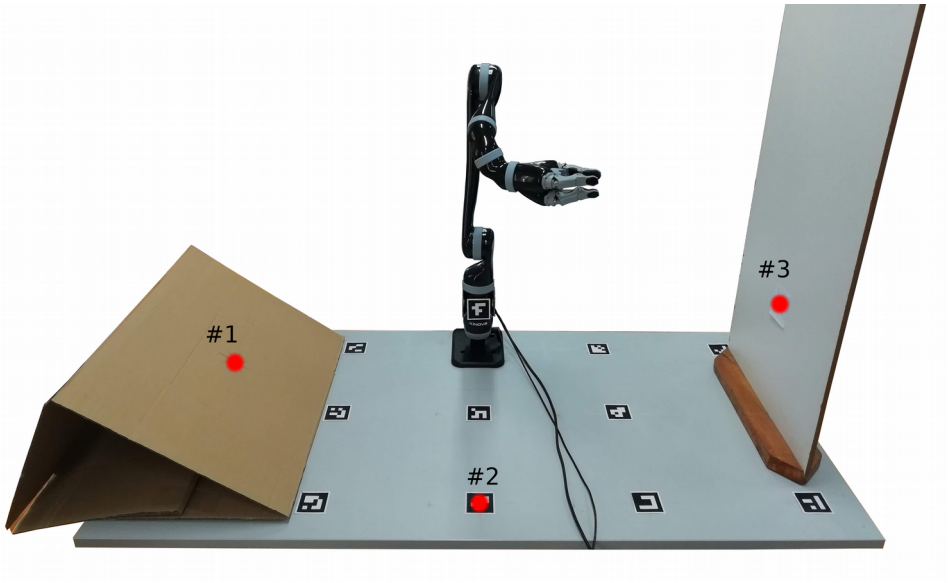


Figure 31: Photo of the arrangement for the final demo of the gaze intersection pose generation method. Sequence of locations used for the final results marked in red.

Important to note that the results displayed use the pose's Z axis vector as an orientation indicator, since the rotation in the Z axis of the end-effector is redundant, since it is produced in relation to the user world location.

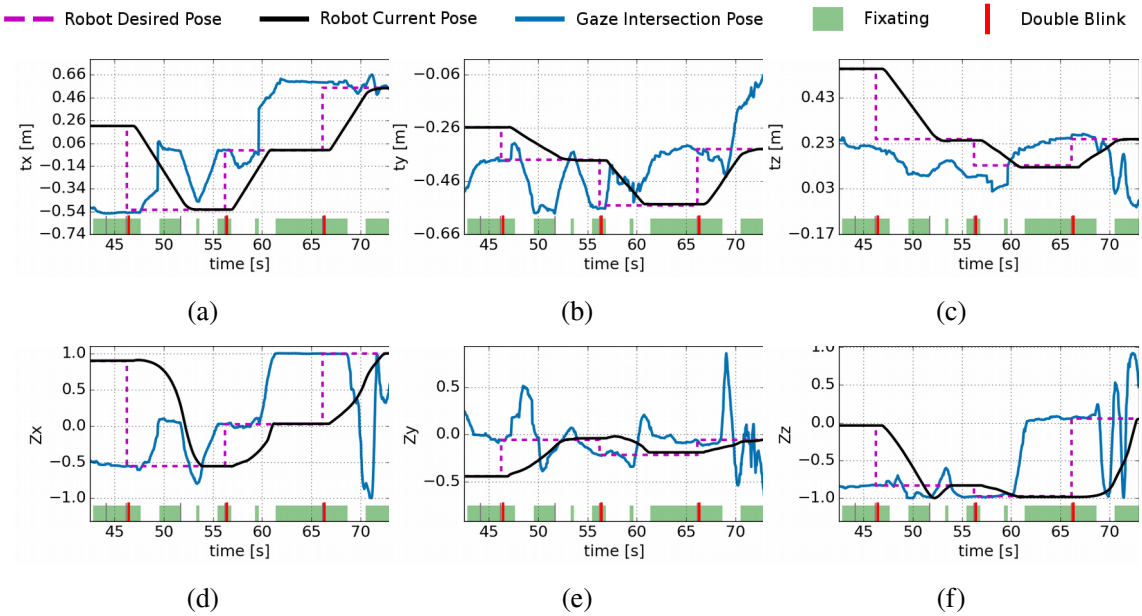


Figure 32: Gaze intersection pose final demo results for 3 predefined locations, includes the robot curves. (a)(b)(c) Position in X, Y and Z world coordinate system axis of the gaze intersection, robot end-effector and robot desired poses. (d)(e)(f) Resulting X, Y and Z components of the intersection, robot end-effector and robot desired poses Z axis vector. With valid fixation period signaled in green and double blink occurance time in vertical solid red line.

It can be noted that, from the current robot pose curves presented in the figure (32), the time span specified to the the robot controller is 5 seconds, this is used internally to interpolated a desired pose trajectory. It is also important to note that the gaze intersection pose is free to move about the workspace completely decoupled from the control loop, though the double blink action signals the system to update the robot goal according to the offseted pose to the gaze intersection pose. Hence in contrast to the previous results seen in (5.1) where the gaze pose should coincide with the expected position, in this demo the position is supposed to have an offset depending on the respective pose orientation. This can mostly be seen in the Z axis position curves in the figure (32).

6.2 Closest Marker Pose

Now, similar to the results obtained from the previous section (5.2), the robot goal pose will be controlled by the eye gaze, however in this module the resulting robot pose is a pose predefined for each of the world static markers and for the marker group placed on the mobile box.

To demonstrate the final results, including the previously excluded mobile box, the user is tasked with a sequence of movements. This task mimics a common pick-and-place where the user is required to move the marked box to one of the static markers locations, which results in following actions:

- Focus on the box and signal the system to move to the corresponding desired pose, in relation to the current box pose;
- Robot control using the joystick to grab the box;
- Focus on a static world markers and signal the system to the corresponding pose, while end-effector is grabbing the aforementioned box.

For testing purposes the box is placed in a known location which corresponds approximately to a desired pose with (0.3, -0.35, 0.21) position and (1, 0, 0) orientation Z axis vector.

Considering that the chosen marker is the marker identified with ***#7*** in (27) and (28) then the desired pose corresponds to the position (0,-0.6, 0.2) and orientation vector of (0, 0, -1).

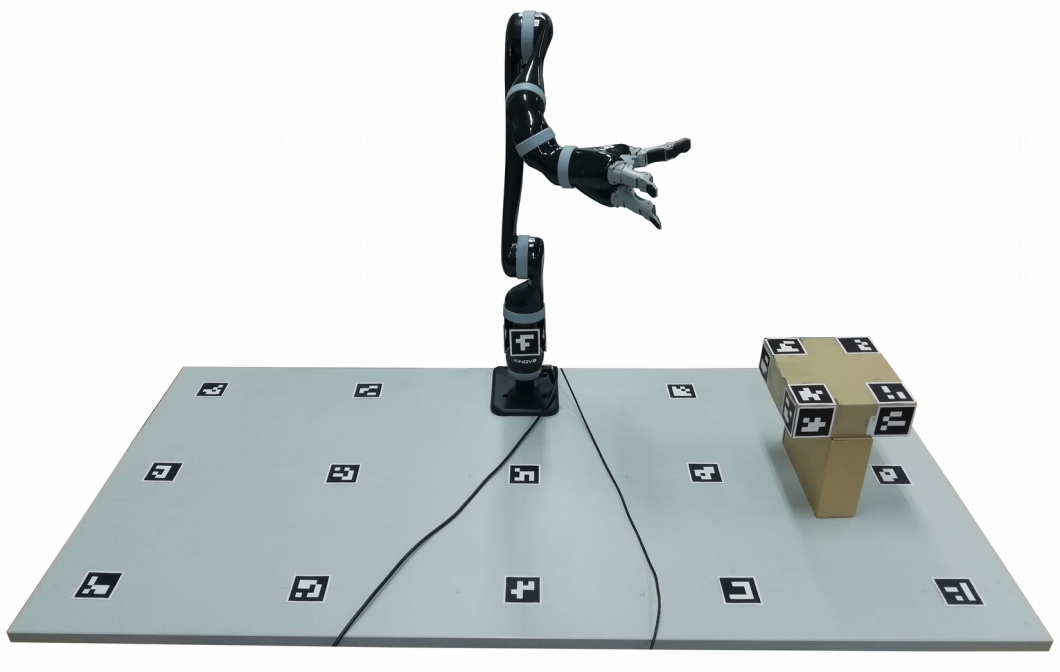


Figure 33: Photo of the arrangement for the final demo of the closest marker pose generation method.

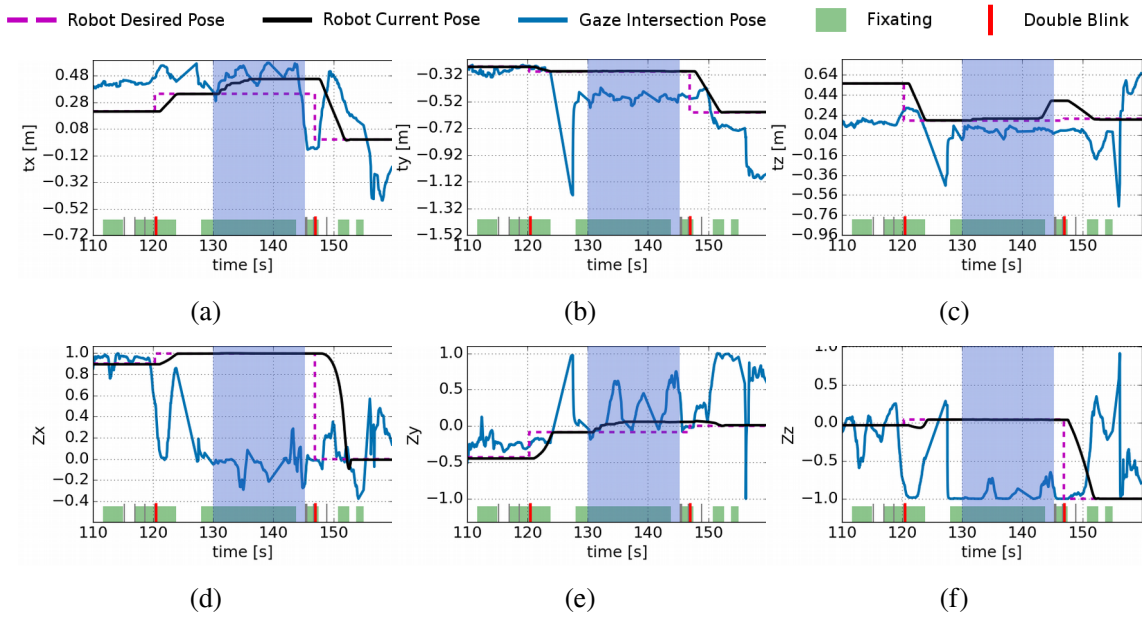


Figure 34: Closest marker final demo results for a pick-and-place action, includes the robot curves. (a)(b)(c) Position in X, Y and Z world coordinate system axis of the gaze intersection, robot end-effector and robot desired poses. (d)(e)(f) Resulting X, Y and Z components of the intersection, robot end-effector and robot desired poses Z axis vector. With valid fixation period signaled in green and double blink occurance time in vertical solid red line.

The previous curve demonstrate the correct detection of the predefined pose for the box, according to the its expected physical location. From the figure it can also be visualized the user manipulation, blue shaded area, where the motion to grab the box is

executed, after the system placement of the robot pose into a desirable pose for such operation. After the motion grabbing motion is completed and the box is manually moved higher, to lower the probability of collisions, after which the user inputs a new pose through the system, placing the box into the desired final pose.

As a baseline the same experiment is also executed using only joystick inputs, manual telemanipulation, resulting in the figure (35) where is can be observed the corresponding manual approach takes longer to complete, around ~60 seconds averaged from multiple tests and users, while the mixed operation takes ~40 seconds. Considering that the joystick only has 3 simultaneous degrees of control then it is necessary to change between control modes to access the 6 degrees of freedom and the additional end-effector closure control. This switching operation increases the required time and complexity of each operation.

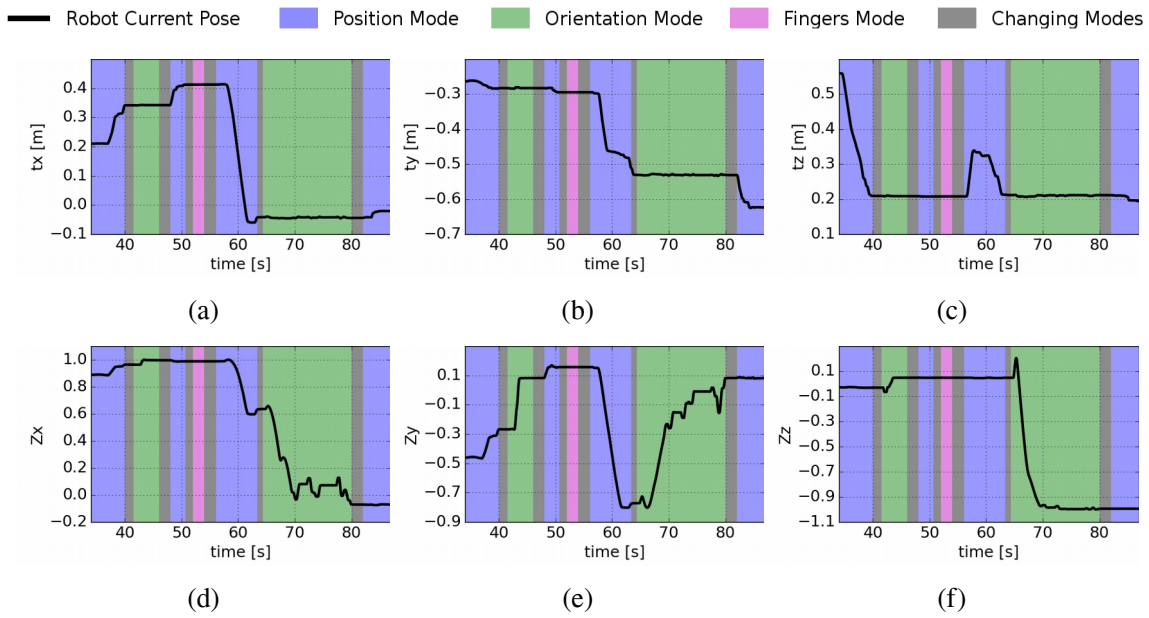


Figure 35: Manual manipulation version of pick-and-place demo. (a)(b)(c) Position in X, Y and Z world coordinate system axis of the robot end-effector pose. (d)(e)(f) Resulting X, Y and Z components of the robot end-effector pose Z axis vector. With visualization of the current actions and modes while operating the robot.

Comparing just manual manipulation curves with the previous mixed manipulation, the resulting time benefits become obvious. This is prevalent considering the wasted time switching between the different modes to achieved the same result.

7 Conclusion

Reviewing the results, it is evident that this type of system has several limitations, especially considering the heavy dependence on a correct detection of both the world and/or the box markers as a prerequisite to the correct function for all of the detailed methods. This of course removes the most important particularity of the system, which is the mobility of the eye tracker by confining it to a predefined and limited workspace.

Taking into consideration that the control methods detailed are directed to the assistive field, such as users with mobility disabilities, then these can be evaluated from an functional point of view.

In regard to the gaze intersection pose method, the resulting accuracy is higher than expected. It is important to highlight that the gaze pose is generated from a gaze vector with an associated error which is further propagated by the errors from the eye tracker pose estimation process and the intrinsic noise of the depth camera. The results obtained validate the potential of this or derived control systems for some applications, however this method functionality is extremely limited, considering the robot used, since most objects with suitable size, for correct orientation estimation of the goal pose, are too wide and can not be grabbed by this robot's end-effector.

Considering the alternative, the closest marker pose, as a solution to the minimum object size limitation of the previous method, then this method functionality is higher. However, since it has the markers detection as a prerequisite, it only works for objects prepared beforehand. However the resulting benefit from this system is clear when considering the difference in the time spent executing the pick-and-place demo between just manual telemanipulation and mixed manipulation, respectively ~60 and ~40 seconds.

The comparison between these control systems and manual manipulation is of course biased. Since these compare a specifically design system with the generic manual manipulation which has a unrestricted functionality. Nevertheless, the obtained results can serve as a validation to this type of gaze guided control systems, and are compelling enough to warrant further research.

7.1 Future Work

Since the base of the system is completed in this dissertation, most of the future work resolves around the addition of extra functionality or the resolution of the problems found during the aforementioned processes. These problems are mainly the high computation cost, which results in low FPS, and the required preparation of the workspace for the methods developed to work. From this the resulting suggestions are:

- Single marker orientation for the box, possible using the Aruco Library, however the use of several markers is still advised, using the excess information to refine the resulting orientation, reducing the possibility of errors and the effect of occlusions.
- Use of a SLAM algorithm for the estimation of the pose of Pupil. Since most of the computation cost is derived from the use of the Aruco Library functions its removal would result in higher FPS. However this SLAM approach should use only the depth stream, since this is not distorted and has higher frame rate, 90 FPS instead of the RGB 30 FPS.
- The use of and object detection system, such as the YOLOv3 which is based on neural networks and as real-time capable, to obtain the portions of the point cloud which represent each of the detected objects. Finally the use of this portions and 3D models of the respective objects to estimate the object orientation in the world. Each object should also have a grasping motion in case they are selected using a similar approach to this dissertation.

BIBLIOGRAPHY

[1] M. F. Washburn and E. B. Huey, "The Psychology and Pedagogy of Reading.," *Philos. Rev.*, 2006.

[2] D. D. Salvucci and J. H. Goldberg, "Identifying Fixations and Saccades in Eye-tracking Protocols," in *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, New York, NY, USA, 2000, pp. 71–78.

[3] J. Pelz, M. Hayhoe, and R. Loeber, "The coordination of eye, head, and hand movements in a natural task," *Exp. Brain Res.*, 2003.

[4] M. Hayhoe and D. Ballard, *Eye movements in natural behavior*. 2005.

[5] H. Admoni, T. Santini, T. C. Kübler, S. Srinivasa, E. Kasneci, and R. M. Aronson, "Eye-Hand Behavior in Human-Robot Shared Manipulation," 2018.

[6] M. K. Eckstein, B. Guerra-Carrillo, A. T. Miller Singley, and S. A. Bunge, *Beyond eye gaze: What else can eyetracking reveal about cognition and cognitive development?* 2017.

[7] M. L. Mele and S. Federici, "Gaze and eye-tracking solutions for psychological research," *Cogn. Process.*, vol. 13, no. S1, pp. 261–265, Aug. 2012.

[8] C. Valuch, L. S. Pflüger, B. Wallner, B. Laeng, and U. Ansorge, "Using eye tracking to test for individual differences in attention to attractive faces," *Front. Psychol.*, 2015.

[9] G. Ziv, "Gaze Behavior and Visual Attention: A Review of Eye Tracking Studies in Aviation," *Int. J. Aviat. Psychol.*, vol. 26, no. 3–4, pp. 75–104, Oct. 2016.

[10] R. Kredel, C. Vater, A. Klostermann, and E. J. Hossner, "Eye-tracking technology and the dynamics of natural gaze behavior in sports: A systematic review of 40 years of research," *Front. Psychol.*, 2017.

[11] R. M. H. S., L. Herlant, H. Admoni, and Siddhartha, "Visibility Optimization in Manipulation Tasks for a Wheelchair-Mounted Robot Arm," 2016.

[12] L. V. Herlant, R. M. Holladay, and S. S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *ACM/IEEE International Conference on Human-Robot Interaction*, 2016.

[13] H. Aronson, Reuben M. and Admoni, "Gaze for Error Detection During Human-Robot Shared Manipulation," 2018.

[14] H. Admoni and S. Srinivasa, "Predicting user intent through eye gaze for shared autonomy," 2016.

[15] S. Javdani, H. Admoni, S. Pellegrinelli, S. S. Srinivasa, and J. A. Bagnell, *Shared autonomy via hindsight optimization for teleoperation and teaming*. 2018.

[16] T. Piumsomboon, G. Lee, R. W. Lindeman, and M. Billinghurst, "Exploring natural eye-gaze-based interaction for immersive virtual reality," in *2017 IEEE Symposium on 3D User Interfaces, 3DUI 2017 - Proceedings*, 2017.

[17] "Tobbi Pro Glasses 2." [Online]. Available: https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/.

[18] "Pupil Labs." [Online]. Available: https://pupil-labs.com/.

[19] J. J. MacInnes, S. Iqbal, J. Pearson, and E. N. Johnson, "Wearable Eye-tracking for Research: Automated dynamic gaze mapping and accuracy/precision comparisons across devices," *bioRxiv*, 2018.

[20] A. B, L. Hanna, K. B. N. Pavan, and Y. Chai, "Calibration Techniques and Gaze Accuracy Estimation in Pupil Labs Eye Tracker," *TECHART J. Arts Imaging Sci.*, 2018.

[21] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, “Intel(R) RealSense(TM) Stereoscopic Depth Cameras,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017.
[22] L. Świrski and N. A. Dodgson, “A fully-automatic, temporal approach to single camera, glint-free 3D eye model fitting,” Pervasive Eye Track. Mob. Eye-Based Interact., no. July, pp. 1–37, 2013.
[23] B. K. P. Horn, “Closed-form solution of absolute orientation using unit quaternions,” J. Opt. Soc. Am. A, 2008.
[24] “Kinova Robotics” [Online]. Available: https://www.kinovarobotics.com/.

A ZEROMQ

ZeroMQ is a network library, allowing various transport types such as intra-process, inter-process, TCP and multicast. From the ZeroMQ documentation: *"We took a normal TCP socket, injected it with a mix of radioactive isotopes stolen from a secret Soviet atomic research project, bombarded it with 1950-era cosmic rays, and put it into the hands of a drug-addled comic book author with a badly-disguised fetish for bulging muscles clad in spandex. Yes, ZeroMQ sockets are the world-saving superheroes of the networking world."*

This library library allows for several communication paradigms such as one-to-many, many-to-one, or many-to-many. This generic asynchronous I/O model provides a concurrency framework base for scalable multicore applications.

ZeroMQ library most basic communications are represented by a socket pair, REQ-REP, which is is a one-to-one, client-to-server, with request-reply pattern, and the PUB-SUB pattern where the subscriber is constantly waiting new messages. The latter is the basis of the implementation in Pupil Software, where the communication backbone is a server which receives messages relays said information in a one-to-many pattern to the available subscribers.

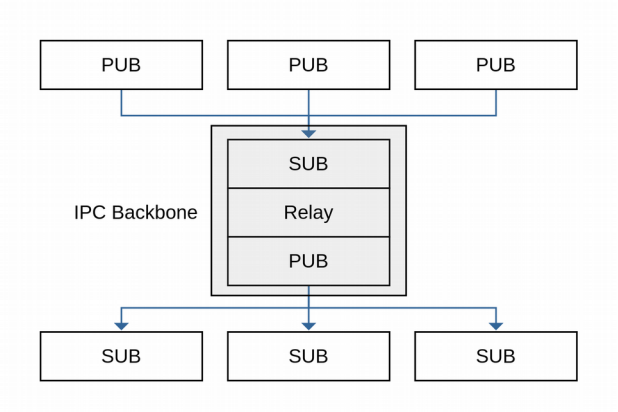


Figure 36: Pupil Software IPC structure using ZeroMQ library.

The use of this library allows, most importantly, for asynchronous communication, automatic dropping of non consumed messages, garanted delivery of the complete message or dropping of said message if this is not possible and automatic repair of broken connections.

B MsgPack

MsgPack is an efficient binary serialization format. This allows simple and compact serialization of complex data structures without lost of its structure, it is not required to have knowledge in advance of the data structure for its correct parsing since the data tags compressed along side the actual data. The official implementation of this library has also been ported to a variety of different programming languages.

This library has several methods for either *pack()* and *unpack()* data structures, considering that only the latter is required then the conversion process from the bytes data receive using the ZeroMQ Library can be unpacked resulting in a *msgpack::object*, which has list of tupples with the each of the compressed variables tag names and actual data. From this *msgpack::object* the conversion to the desired custom type requires the implementation of a *convert()* method .

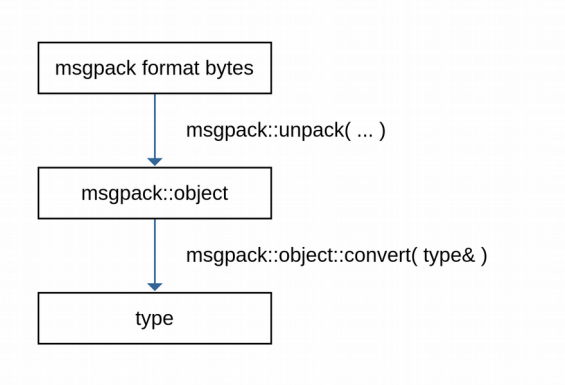


Figure 37: Conversion process from bytes to custom types, using msgpack library.

Example of the implementation of the non-intrusive convert() method

```
namespace msgpack {
MSGPACK_API_VERSION_NAMESPACE(MSGPACK_DEFAULT_API_NS) {
namespace adaptor {

template<>
struct convert<type> {
    msgpack::object const& operator()(msgpack::object const& o, type& msg)const{
        // Parsing the msgpack::object into the custom type msg

        return o;
    }
};
}}}
```

C ROS OVERVIEW

ROS is an open-source, meta-operating system direct to robotic applications. Ir provides services expected in an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionalities, inter-process message transmission, and package management.

The ROS runtime is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. This includes implementations of different styles of communication such as, RPC-style using the ROS services, asynchronous streaming of data using topics, and data storage on a global ROS parameter server. Important to note that ROS is not a realtime framework.

The ROS can be divided mainly in the following components:

Nodes – Modular processes for computation, these can be seen as functions in which the inputs are the messages to which each node has subscribed;

Master – The actual server which takes has a list of all the current ROS components in use and takes care of the connection between each of the components as required, it provides lookup information for the nodes atempting communication, similar to a DNS server.

Messages – Data structures passed from node to node, can either be generic ROS messages, or custom generated messages;

Parameter Server – Global data storage of key elements of run-time configurations for the nodes;

Topics – Name identifying the origin or destination of the ROS messages, these can be used in a Many-to-Many communication paradigm. A node can publish a message to a specific topic, and likewise, a node can also receive a message from said topic. Only the Master has knowledge of the run-time topics, and a connection request is required for nodes to use a specific topic name for communication.

Services – Request/reply communication, used as alternative to the ROS Topics.

Since the ROS is simply a server connecting different processes, then each node has to copy/fill the desires messages with data before publishing, which means a undesired copy information that can, especially in high throughpu data, become a limiting factor. To resolve this issue a ROS package, **nodelets**, was introduced. These add an additional node type where the previous nodes are meshed together into the same process eliminating the required message copy step. These require only a specific code structure and knowledge of the name of the parent process/nodelet and are highly advised for image processing.

A nodelet is simply a base class which has the necessary methods for intra-process communication by spawning threads which runs a specific *init()* method, from which is possible to request connection to other nodes/nodelets, through the ROS topics, and iterate through each of the respective callbacks.

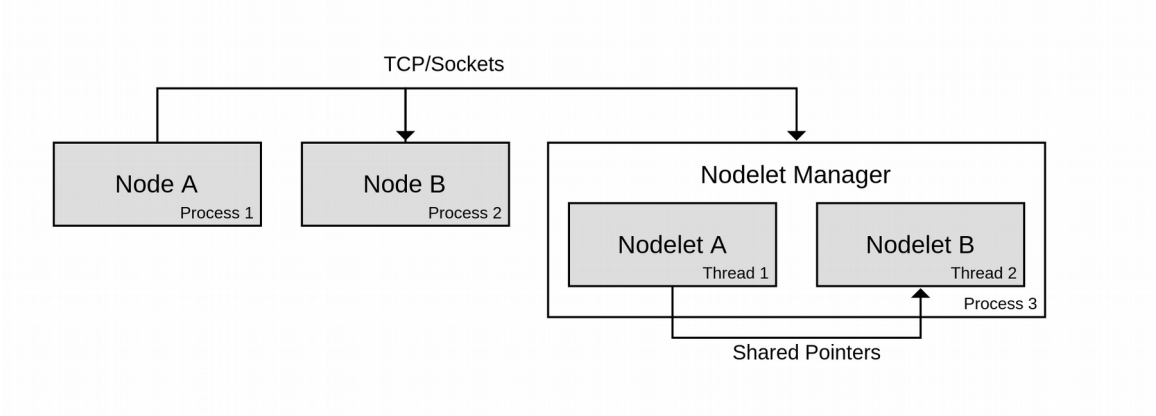


Figure 38: Example of ROS communication structure using both nodes and nodelets.

Example of the implementation of a nodelet class

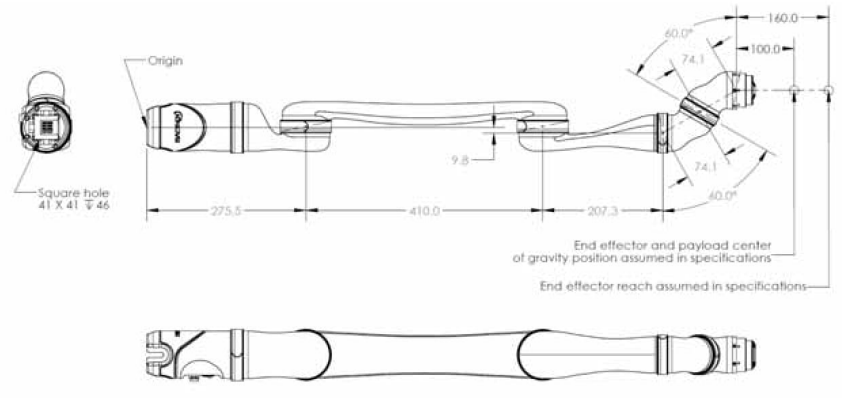
```
namespace package_name {

class NodeletName : public nodelet::Nodelet {
private:
    ros::Subscriber sub_;
public:
    NodeletName(){}
    ~NodeletName(){}

    virtual void onInit() {
        ros::NodeHandle nh = getNodeHandle();
        sub_ = nh.subscribe("/topic_name", 1,
                            package_name::NodeletName::callbackFun, this);
    }

    void callbackFun(/* message_type */) {
        // Process received message
    }
};
}
PLUGINLIB_EXPORT_CLASS(package_name::NodeletName, nodelet::Nodelet);
```

D KINOVA JACO²

Tech Specs

JACO²
6 DOF

Version 1.1 – April 2017



GENERAL

		NO GRIPPER	2 FINGERS (KG-2)	3 FINGERS (KG-3)
Total weight		4.4 kg	5.0 kg	5.2 kg
Payload capabilities	Mid-range continuous	2.6 kg	1.8 kg	1.6 kg
	Full-reach peak/temporary	2.2 kg	1.5 kg	1.3 kg
Materials	Links	Carbon fiber		
	Actuators	Aluminum		
Maximum reach		90 cm		
Joint range after start-up *(sotware limitation)*		±27.7 turns		
Maximum linear arm speed		20 cm/s		
Power supply voltage		18 to 29 VDC, 24 VDC nominal		
Peak power		100 W		
Average power	Operating mode	25 W		
	Standby mode	5 W		
Communication protocol		RS-485		
Communication cables		20 pins flat flex cable		
Expansion pins		2 *(on communication bus)*		
Water resistance		IPX2		
Operating temperature		-10 °C to 40 °C		

CONTROLLER

Ports	Joystick	1 Mbps Canbus
	Power supply	18 to 29 VDC, 24 VDC nominal
	USB 2.0 (API)	12 Mbps
	Ethernet (API)	100 Mbps
Control system frequency	High level (API)	100 Hz
	Low level (API)	500 Hz
CPU		360 MHz
SDK	APIs	High and low level
	Compatibility	Windows, Linux Ubuntu & ROS
	Port	USB 2.0, Ethernet
	Programming languages	C++
Control		Force, cartesian & angular

SPECIFICATIONS

Actuators #1, #2 & #3	K-75+
Actuators #4, #5 & #6	K-58

KINOVA ROBOTICS
4333 Grande-Allée Boulevard,
Boisbriand (Québec) J7H 1M7
info@kinovarobotics.com

1-855-6-KINOVA kinovarobotics.com

JACO² is a product of Kinova Robotics, designed and manufactured in Canada.

E INSTALLATION REQUIREMENTS

- Installation of the Linux 16.04;
- Installation of the required kernel version;

```
$ sudo apt-get install linux-image-generic-lts-xenial
$ sudo reboot  # Choose kernel version 4.4.0-135-generic in boot menu
```

- Installation of ROS Kinetic, as per online instructions;
- Installation of the librealsense;

```
$ sudo apt install libusb-1.0-0-dev pkg-config libglfw3-dev
$ cd ~/ && git clone https://github.com/IntelRealSense/librealsense.git
$ cd librealsense && git checkout v1.12.1
$ mkdir build && cd build
$ cmake .. -DBUILD_EXAMPLES:BOOL=false
$ make -j4 && sudo make install
```

- Patch linux kernel;

```
$ sudo cp config/99-realsense-libusb.rules /etc/udev/rules.d/
$ sudo udevadm control --reload-rules && udevadm trigger
$ sudo apt-get install libusb-1.0-0-dev
$ sudo apt-get install linux-headers-generic build-essential
$ git clone git://kernel.ubuntu.com/ubuntu/ubuntu-xenial.git --depth 1
$ cd ubuntu-xenial
$ sudo patch -p1 < ../scripts/realsense-camera-formats_ubuntu16.patch
$ sudo cp debian/scripts/retpoline-extract-one scripts/ubuntu-retpoline-extract-
one
$ cd .. && sudo ./scripts/patch-uvcvideo-16.04.simple.sh
```

- Checking correct Intel Realsense camera detection;

  Before connecting the camera usb, clean the kernel log.

```
$ sudo dmesg --clear
```

  Connect the camera and check the kernel log. Output should be similar to:

```
$ dmesg
[   12.933984] usb 2-1.3: new SuperSpeed USB device number 5 using xhci_hcd
[   12.952802] usb 2-1.3: New USB device found, idVendor=8086, idProduct=0a80
[   12.952806] usb 2-1.3: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[   12.952809] usb 2-1.3: Product: Intel RealSense 3D Camera R200
[   12.952811] usb 2-1.3: Manufacturer: Intel Corp
[   12.952812] usb 2-1.3: SerialNumber: SN_2461010333
```

  Where it is possible to see correct detection of the camera information. Similar approach with:

```
$ usb-devices | grep Product
(…)
S:  Product=Intel RealSense 3D Camera R200
(…)
```

F Pick and Place Demo Photos

Figure 39: Video frames of the Pick and Place demo manual telemanipulation.

Figure 40: Video drames of the Pick and Place demo mixed manipulation.