EE492 Senior Project

EE492 Senior Design Project Final Report
Multi-Robot Realization

BERKAY GÜMÜŞ

Department of Electrical and Electronic Engineering

Boğaziçi University

Bebek, Istanbul 34342

Project Advisor: H. Işıl Bozma

Evaluation Committee Member: Yağmur Denizhan

Evaluation Committee Member: Mehmet Akar

July 11, 2020

Acknowledgements

I would like to express my special thanks of gratitude to my project advisor Professor H. Işıl Bozma for giving me this project and working with her expert advice and encouragement opportunity at ISL for 2 years. Furthermore, I would like to express my appreciation to my graduate advisor Kadir Türksoy for his help and suggestions.

Abstract

This report is about the second part of the EE491-492 project. This project considers multi-robot formation by using the stereo camera system on the robots. The main task of the project is to make minik robots randomly located be able to realize any given form identified by an adjacency matrix by taking position information from the stereo camera systems.

Contents

List of Figures

1 Introduction

In this project, we are concerned with multi-robot realization problem. In this problem, the robots need to attain a goal topology. This is an important problem in many multi-robot applications. For example, the application may require the robots to be in a line as in the case in search and rescue or robotic soccer. Realizing a goal topology is relatively easy with small number of robots such as two. However, real-life scenarios require much more robots to execute the task.

An inexact solution to the realization problem has been presented in [3]. In this approach, the goal topology is specified by an adjacency matrix and an adjacency threshold. Furthermore, the robots need to know their distances to the other goals as they are moving - as expressed by a distance matrix. The corresponding control laws are automatically generated and then given as input to the robots.

This project focuses on the practical implementation of multi-robot realization. The robots that is used in the project are Minik robots. Minik robots are small sized differential type robots with RaspberryPi 3 processor, an Arduino Uno to drive electric motors and a stereo camera with two webcams as shown in Fig. 1a. For the distance matrix, the stereo camera systems on the robots are used as a sensor as shown in Fig. 1b.

(a) Minik Robots

(b) Stereo Camera

Figure 1: Minik Robots and the Camera System

2 Problem Statement

The main goals of this project are as follows:

- Realization: The robots will attain a given goal topology as specified by an adjacency matrix A and an adjacency threshold $\rho > 0$. The goal is to have the robots move in a such way as to attain the given topology - namely

$$\delta_{ij} = \begin{cases} <= \rho & \text{if } a_{ij} = 1 \\ > \rho & \text{if } a_{ij} = 0 \end{cases} \quad (1)$$

  The proposed approach is presented in [3].

- Vision-based Distance Matrix Computation: The robots need to collectively compute the distance matrix as they are moving to attain the goal topology. This is based on the information that is obtained from the the stereo camera system on the robots.

3 Method

This is a two-term project whose first half -first semester- consists of implementation of the robot movement algorithm in the article. The first part is completed and tested using over-head camera system. The first part can be checked at 491 Final Report.
The second part consists of using this realization algorithm with stereo camera systems on the robots rather than over-head camera system. Over-head camera system is robust and calculates orientation and position of each robot at any time with respect to world frame (over-head camera frame). These positions and orientations can be used at the realization algorithm directly. However, stereo camera systems calculates robot positions with respect to their own frames and these frames must be transformed to be able to use at the realization algorithm and the position data taken stereo camera system is not accurate as the position data taken over-head camera system.

The project consists of the following stages:

- Operating Minik Robots: Software
- Object Position Calculation with respect to Initial Frame with Stereo Camera
- Initial Robot Frame Transformations
- Object Detection Methods
- Gazebo Implementation

3.1 Operating Minik Robots: Software

Minik robots are consisting of many hardware and software such as ROS packages, arduino as motor controller and raspberry pi3 as processor. This project doesn't contain odometry localization and low level motor controller task but they are closely related to realization. These part must be accurate and stable to make this project. There are some essential packages for the system. These are

- minik_ros_wrapper
- robot_localization_deneme
- minik_keyop

The package minik_ros_wrapper is about low_level motor controller and it provides communication between raspberry pi and arduino using serial port communication. The package sends the velocity commands to arduino and takes the velocity feedback from arduino.

The package robot_localization_deneme is about localization. It was using only odometry information, there was not any extra sensor. It estimates the position of the Minik using velocity feedback taken from motor controller and publishes it at the ROS workspace to be used by other packages such realization packages.

The package minik_keyop is about the manual control. It's like a remote controller. Sometimes, navigation codes have bugs and the robot goes to wrong places and this situation can be risky. At these situations, the robot must be stopped.

These packages are communicating each other via ROS topics and these ROS topics must have proper name, proper type and proper data. One package publishes a topic whereas another package subscribes this topic. So, the name and the type of these topics which is declared at the packages must be same to be able to communicate. I investigated their names and types and made them proper and understandable.

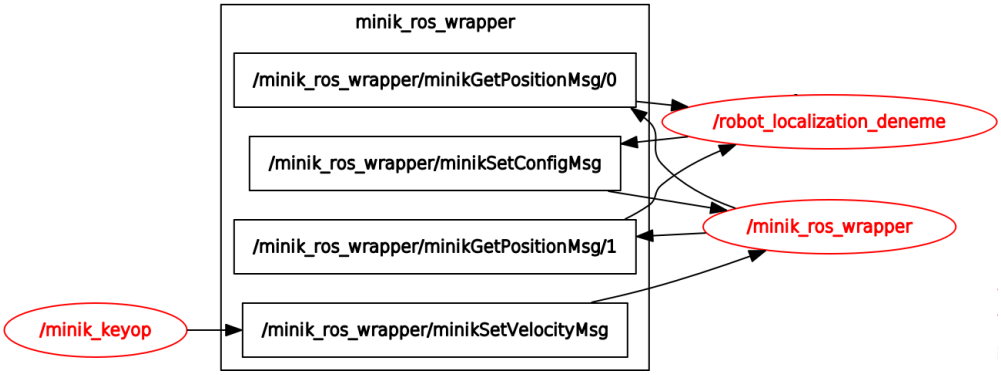


Figure 2: Minik ROS Package Architecture without Realization and Stereo Camera System

3.2 Object Position Calculation with respect to Initial Frame with Stereo Camera System

The calibration, object detection and distance calculation parts are completed at the first term and explained at EE491 Final Report. This subsection is related to object position with respect to initial robot frame.

- Let $F^i(t)$ be the frame of robot i and $F^{ci}(t)$ be the frame of camera system on the robot i at time t.
- Let $P^i_j(t) \in R^2$ be the position of robot j with respect to the frame $F^i(0)$ at time t.
- Let $P^i_i(t) \in R^2$ be position of robot i with respect to $F^i(0)$.
- Let $P^{ci}_j(t) \in R^2$ be position of robot j with respect to $F^{ci}(t)$.
- Let $\theta_{ici} \in S^1$ be the angle (heading of the robot i) and R^i_{ci} be rotation matrix from $F^i(0)$ to frame $F^{ci}(t)$.

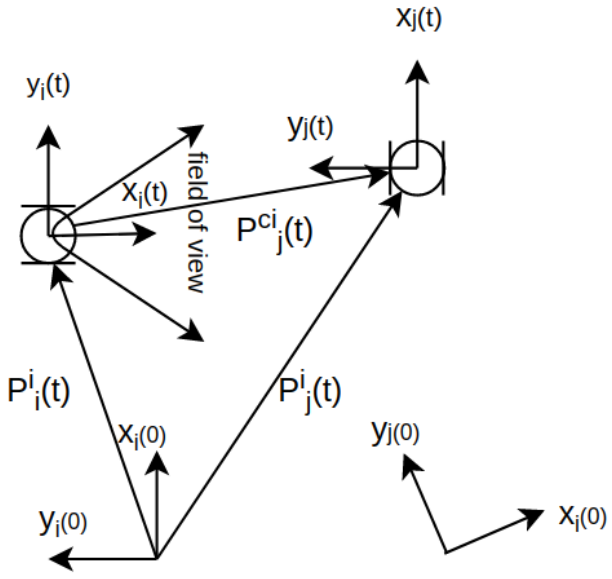


Figure 3: Stereo Camera and Robot Frames

The odometry system calculates $P_i^i(t)$ and heading of the robot i at any time t and stereo camera system calculates $P_j^{ci}(t)$ at time t when the robot i observes robot j.

$$
\begin{aligned}
P_j^i(t) &= P_i^i(t) + R_{ci}^i P_j^{ci}(t) \\
\begin{bmatrix} P_j^i x(t) \\ P_j^i y(t) \end{bmatrix} &= \begin{bmatrix} P_i^i x(t) \\ P_i^i y(t) \end{bmatrix} + \begin{bmatrix} cos(\theta_{ici}) & -sin(\theta_{ici}) \\ sin(\theta_{ici}) & cos(\theta_{ici}) \end{bmatrix} \begin{bmatrix} P_j^{ci} x(t) \\ P_j^{ci} y(t) \end{bmatrix}
\end{aligned} \tag{2}
$$

The stereo camera system is implemented at camera2ros package. It calculates and publish the robot positions observed.

3.3 Initial Robot Frame Transformations

Let there be n robots. These robots are capable of calculating of the objects with specific colors using stereo camera systems. The goal is to make minik robots measure the positions of each others with respect to their frames.

Robots can measure their own positions and orientations using the odometry data any time; and the positions of other robots at the field of views with respect to their own initial frames using the stereo camera vision system. However, the realization algorithm needs each positions with respect to each frame. So, we develop 2 methods related to the frame transformations to calculate positions even if the robots are not at the view points. After finding initial frame transformations, the robots can calculate the positions of other robots converting the positions at other frames to their own frames.

- Let $F^i(t)$ be the frame of robot i at time t.
- Let $P_j^i(t) \in R^2$ be the position of robot j with respect to the frame $F^i(0)$ at time t.
- Let $P_i^i(t) \in R^2$ be position of robot i with respect to $F^i(0)$.
- Let d_j^i be the position of $F^i(0)$ with respect to $F^j(0)$.
- Let $\theta_{ij} \in S^1$ be the angle and R_j^i be rotation matrix from $F^i(0)$ to frame $F^j(0)$.

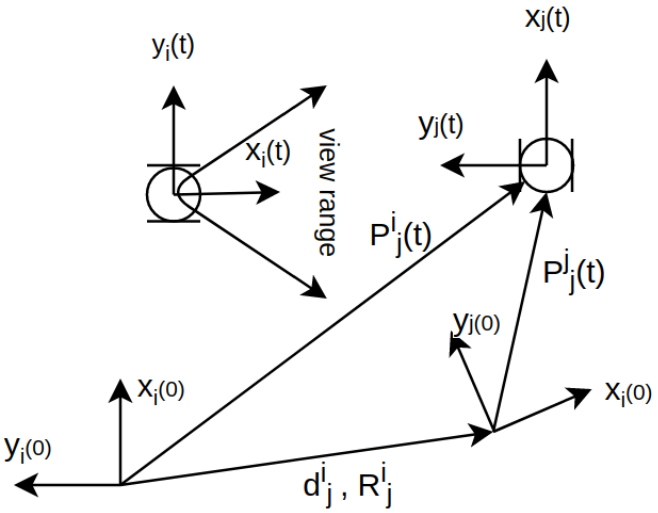


Figure 4: Robot Frames

3.3.1 Method 1

Each robot moves and suppose robot i observes robot j at time t_1 and t_2.
Observation at t_1:

$$\begin{aligned} P_j^i(t_1) &= R_j^i P_j^j(t_1) + d_j^i \\ d_j^i &= P_j^i(t_1) - R_j^i P_j^j(t_1) \end{aligned} \tag{3}$$

Observation at t_2:

$$\begin{aligned} P_j^i(t_2) &= R_j^i P_j^j(t_2) + d_j^i \\ d_j^i &= P_j^i(t_2) - R_j^i P_j^j(t_2) \end{aligned} \tag{4}$$

Using eq. 3 and 4:

$$\begin{aligned} P_j^i(t_2) - R_j^i P_j^j(t_2) &= P_j^i(t_1) - R_j^i P_j^j(t_1) \\ P_j^i(t_2) - P_j^i(t_1) &= R_j^i\big(P_j^j(t_2) - P_j^j(t_1)\big) \end{aligned} \tag{5}$$

$P_j^j(t_2)$ and $P_j^j(t_1)$ are taken from the odometry data whereas $P_j^i(t_2)$ and $P_j^i(t_1)$ are taken from the stereo vision data. Then, the rotation matrix and the angle can be calculated.

$$\begin{bmatrix} P_j^i x(t_2) - P_j^i x(t_1) \\ P_j^i y(t_2) - P_j^i y(t_1) \end{bmatrix} = \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} P_j^j x(t_2) - P_j^j x(t_1) \\ P_j^j y(t_2) - P_j^j y(t_1) \end{bmatrix} \tag{6}$$

Δ^i and Δ^j are the displacement vectors from t_1 to t_2 with respect to frame i and frame j, respectively.

$$\begin{bmatrix} \Delta_x^i \\ \Delta_y^i \end{bmatrix} = \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} \Delta_x^j \\ \Delta_y^j \end{bmatrix} \tag{7}$$

θ^i and θ^j are the angles of position vectors with respect to frame i and frame j, respectively.

$$\begin{aligned} arctan2(\Delta_y^i, \Delta_x^i) &= \theta^i \\ arctan2(\Delta_y^j, \Delta_x^j) &= \theta^j \end{aligned} \tag{8}$$

$$\begin{aligned} \theta_{ij} &= \theta^i - \theta^j \\ \theta_{ji} &= \theta^j - \theta^i \end{aligned} \tag{9}$$

The transformation angles for frame i and frame j are found. Then, transformation positions (d_j^i and d_i^j) can be calculated using eq. 4.

$$\begin{aligned} d_j^i &= P_j^i(t_2) - R_j^i P_j^j(t_2) \\ \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} &= \begin{bmatrix} P_j^i x(t_2) \\ P_j^i y(t_2) \end{bmatrix} - \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} P_j^j x(t_2) \\ P_j^j y(t_2) \end{bmatrix} \end{aligned} \tag{10}$$

Also,

$$\begin{aligned} d_i^j &= -R_i^j d_j^i \\ \begin{bmatrix} d_i^j x \\ d_i^j y \end{bmatrix} &= \begin{bmatrix} cos(-\theta_{ij}) & -sin(-\theta_{ij}) \\ sin(-\theta_{ij}) & cos(-\theta_{ij}) \end{bmatrix} \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} \\ \begin{bmatrix} d_i^j x \\ d_i^j y \end{bmatrix} &= \begin{bmatrix} cos(\theta_{ji}) & -sin(\theta_{ji}) \\ sin(\theta_{ji}) & cos(\theta_{ji}) \end{bmatrix} \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} \end{aligned} \tag{11}$$

Transformation positions for frame i and frame j are found.

3.3.2 Method 2

Each robot moves and suppose robot i observes robot j at time t_1 and robot j observes robot i at time t_2. Observation at t_1:

$$\begin{aligned} P_j^i(t_1) &= d_j^i + R_j^i P_j^j(t_1) \\ d_j^i &= P_j^i(t_1) - R_j^i P_j^j(t_1) \end{aligned} \tag{12}$$

Observation at t_2:

$$\begin{aligned} P_i^j(t_2) &= d_i^j + R_i^j P_i^i(t_2) \\ &= -R_i^j d_j^i + R_i^j P_i^i(t_2) \\ &= -R_i^j (d_j^i - P_i^i(t_2)) \end{aligned} \tag{13}$$

Using eq. 13

$$\begin{aligned} -R_j^i P_i^j(t_2) &= d_j^i - P_i^i(t_2) \\ d_j^i &= P_i^i(t_2) - R_j^i P_i^j(t_2) \end{aligned} \tag{14}$$

Using eq. 12 and 14:

$$\begin{aligned} P_i^i(t_2) - R_j^i P_i^j(t_2) &= P_j^i(t_1) - R_j^i P_j^j(t_1) \\ P_i^i(t_2) - P_j^i(t_1) &= R_j^i P_i^j(t_2) - R_j^i P_j^j(t_1) \\ P_i^i(t_2) - P_j^i(t_1) &= R_j^i \left(P_i^j(t_2) - P_j^j(t_1)\right) \end{aligned} \tag{15}$$

$P_i^i(t_2)$ and $P_j^j(t_1)$ are taken from the odometry data whereas $P_i^j(t_2)$ and $P_j^i(t_1)$ are taken from the stereo vision data. Then, the rotation matrix and the angle can be calculated.

$$\begin{bmatrix} P_i^i x(t_2) - P_j^i x(t_1) \\ P_i^i y(t_2) - P_j^i y(t_1) \end{bmatrix} = \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} P_i^j x(t_2) - P_j^j x(t_1) \\ P_i^j y(t_2) - P_j^j y(t_1) \end{bmatrix} \tag{16}$$

Δ^i and Δ^j are the displacement vectors from t_1 to t_2 with respect to frame i and frame j, respectively.

$$\begin{bmatrix} \Delta_x^i \\ \Delta_y^i \end{bmatrix} = \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} \Delta_x^j \\ \Delta_y^j \end{bmatrix} \tag{17}$$

θ^i and θ^j are the angles of position vectors with respect to frame i and frame j, respectively.

$$\begin{aligned} arctan2(\Delta_y^i, \Delta_x^i) &= \theta^i \\ arctan2(\Delta_y^j, \Delta_x^j) &= \theta^j \end{aligned} \tag{18}$$

$$\begin{aligned} \theta_{ij} &= \theta^i - \theta^j \\ \theta_{ji} &= \theta^j - \theta^i \end{aligned} \tag{19}$$

The transformation angles for frame i and frame j are found. Then, transformation positions (d_j^i and d_i^j) can be calculated using eq. 12.

$$\begin{aligned} d_j^i &= P_j^i(t_1) - R_j^i P_j^j(t_1) \\ \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} &= \begin{bmatrix} P_j^i x(t_1) \\ P_j^i y(t_1) \end{bmatrix} - \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} P_j^j x(t_1) \\ P_j^j y(t_1) \end{bmatrix} \end{aligned} \tag{20}$$

Also,

$$\begin{aligned} d_i^j &= -R_i^j d_j^i \\ \begin{bmatrix} d_i^j x \\ d_i^j y \end{bmatrix} &= \begin{bmatrix} cos(-\theta_{ij}) & -sin(-\theta_{ij}) \\ sin(-\theta_{ij}) & cos(-\theta_{ij}) \end{bmatrix} \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} \\ \begin{bmatrix} d_i^j x \\ d_i^j y \end{bmatrix} &= \begin{bmatrix} cos(\theta_{ji}) & -sin(\theta_{ji}) \\ sin(\theta_{ji}) & cos(\theta_{ji}) \end{bmatrix} \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} \end{aligned} \tag{21}$$

Transformation positions for frame i and frame j are found.

3.3.3 Completing Transformations

If the transformations between frame i and frame j and between frame j and k are known and the transformation between frame i and frame k is unknown, the transformation between frame i and frame k can be calculated without using two methods above.

$$\begin{aligned} \theta_{ik} &= \theta_{ij} + \theta_{jk} \\ \theta_{ki} &= -\theta_{ik} \end{aligned} \tag{22}$$

$$\begin{aligned} d_k^i &= d_j^i + R_j^i d_k^j \\ \begin{bmatrix} d_k^i x \\ d_k^i y \end{bmatrix} &= \begin{bmatrix} d_j^i x \\ d_j^i y \end{bmatrix} + \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} d_k^j x \\ d_k^j y \end{bmatrix} \end{aligned} \tag{23}$$

Also,

$$\begin{aligned} d_i^k &= -R_i^k d_k^i \\ \begin{bmatrix} d_i^k x \\ d_i^k y \end{bmatrix} &= \begin{bmatrix} cos(\theta_{ki}) & -sin(\theta_{ki}) \\ sin(\theta_{ki}) & cos(\theta_{ki}) \end{bmatrix} \begin{bmatrix} d_k^i x \\ d_k^i y \end{bmatrix} \end{aligned} \tag{24}$$

3.3.4 Robot Positions

After calculation of the transformations for the frame i and frame j,the position of robot j at frame i can be calculated any time even if robot i can not see robot j.

$$\begin{aligned} P_j^i(t) &= d_j^i + R_j^i P_j^j(t) \\ \begin{bmatrix} P_j^i x(t) \\ P_j^i y(t) \end{bmatrix} &= \begin{bmatrix} d_j^i x(t) \\ d_j^i y(t) \end{bmatrix} + \begin{bmatrix} cos(\theta_{ij}) & -sin(\theta_{ij}) \\ sin(\theta_{ij}) & cos(\theta_{ij}) \end{bmatrix} \begin{bmatrix} P_j^j x(t) \\ P_j^j y(t) \end{bmatrix} \end{aligned} \tag{25}$$

3.3.5 Modification of Realization

The realization algorithm needs $P_j^i(t)$ for each robot pairs. Without transformations of robot frames, to do this, all robots must be at the fields of view of all other robots. It's not occurred at many cases and the realization algorithm cannot start and work properly. To complete transformations of robot frames, the discovery movement algorithms starts before the realization algorithms. It's quite simple and it guarantees that robots are connected to each other, directly or indirectly and transformations are completed. At the discovery movement algorithms, all robots turn around with an angular velocity and zero linear velocity. Each robot can observe at least one robot (adjacent robot) and this adjacent robot observes this robot. Using method 2 at section 3.3.2, transformations will be completed for each robot frame pairs.

If more than 2 robots are placed at a line at the initial configuration, the robots at the end points can not observe each other and they can be connected indirectly with mid-robots(adjacent robots) using completing method at section 3.3.3.

At the implementation, there are some constraints to increase the accuracy of the transformations:

The data come from the stereo camera system is deleted after 10 seconds. Because, the out of date position data can cause wrong calculations for the transformations due to the error of odometry data. When the time passes and robot moves, some factors such as wheel slippage increase the error of odometry data.

For the method 2, the distance between t1 and t2 must be more than a proper threshold value. This value must be much higher than the error of the odometry and stereo vision systems. However, if it is too high, any robots may not move far enough to use this method.

The transformations are updated when the method 1 or 2 is available. Some of transformations may remain same if there is not available case for method 1 and 2 for these robot pairs.

The missing transformations are fulfilled by the completing method (chapter 3.3.3) before the realization algorithm runs. After the algorithm begins, they can be updated with only method 1 and 2.

3.3.6 Sensor Package for Simulation

We use ROS turtlebot simulator, it takes velocities calculated by the realization algorithm, moves the robots and publish robot positions and orientations with respect to world frame. However, at this realization problem, we do not have positions with respect to world frame. So, we need to convert them to positions and orientations calculated by the odometry and stereo camera systems with respect to each initial robot frame in order to test our transformations.

- Let $F^i(t)$ be the frame of robot i and $F^W(t)$ be the world frame at time t.
- Let $P_j^i(t) \in R^2$ be the position of robot j with respect to the frame $F^i(0)$ at time t.
- Let $P_j^W(t) \in R^2$ be the position of robot j with respect to the frame $F^w(0)$ at time t.
- Let $P_i^i(t) \in R^2$ be position of robot i with respect to $F^i(0)$.
- Let d_j^i be the position of $F^i(0)$ with respect to $F^j(0)$.
- Let $\theta_{ij} \in S^1$ be the angle and R_j^i be rotation matrix from $F^i(0)$ to frame $F^j(0)$.
- Let $\theta_i \in S^1$ be the angle and R_i^W be rotation matrix from $F^W(0)$ to frame $F^i(0)$.

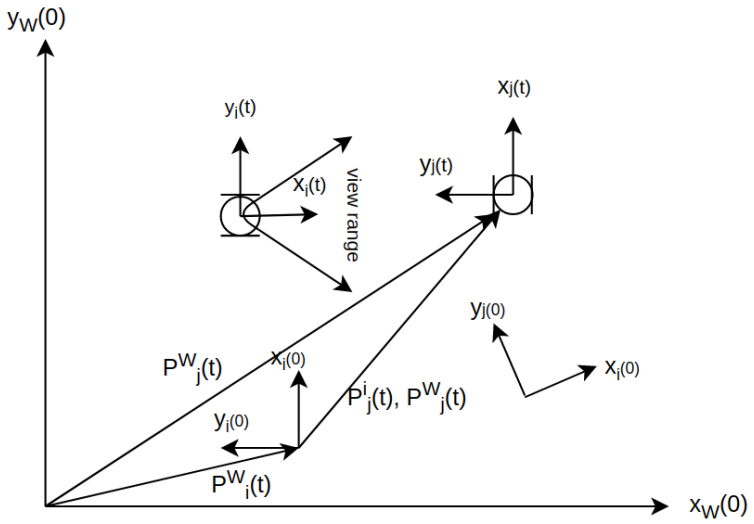


Figure 5: World and Robot Frames

At the realization problem, robot moves and suppose robot i observes robot j. Then, robot i calculates the position of robot j with respect to initial robot i frame ($F^i(0)$). However, at ROS Turtlebot simulator, we know $P_j^W(t)$, $P_j^W(t)$, θ_i and θ_j at any time t.

$$\begin{aligned} P_j^i(t) &= R_W^i(P_j^W(t) - P_i^W(t)) \\ \begin{bmatrix} P_j^i x(t) \\ P_j^i y(t) \end{bmatrix} &= \begin{bmatrix} cos(-\theta_i) & -sin(-\theta_i) \\ sin(-\theta_i) & cos(-\theta_i) \end{bmatrix} \begin{bmatrix} P_j^W x(t) - P_i^W x(t) \\ P_j^W y(t) - P_i^W y(t) \end{bmatrix} \end{aligned} \tag{26}$$

When the algorithm starts, it saves initial positions and orientations with respect to world frames($P_j^W(0)$ and θ_i), takes current positions($P_j^W(t)$) and calculates necessary information ($P_j^i(t)$ and $P_i^i(t)$) similarly camera stereo systems.

After the virtual stereo camera system calculates $P_i^i(t)$ as odometry information and publish at the workspace.

However, in order to publish $P_j^i(t)$, it is necessary to check whether robot i observes robot j at time t for all robot i,robot j pairs. If robot j is at the view point of robot i and it observes, the data is published at the workspace for the realization algorithm to use.

Also, this simulator is completely accurate whereas the real stereo camera systems have some noise.Even if the robot transformations methods work properly with the simulator, it may not work properly and give solid results with real noisy sensors. So, the simulator adds some noise to check results and how the methods work with noisy data.

3.3.7 Package Architecture of Sensor Simulation

There are 3 main custom packages for the simulation, 1 package to check position errors and turtlebot simulation package.

- sensor package
  It subscribes "turtle$i$/pos" topics published by turtlebot simulator package and publishes "pos$i$" for each robot i.
  "turtle$i$/pos" is $P_i^W x(t)$, "odom$i$" is the $P_i^i x(t)$ and "pos$i$" is the position array consisting of $P_j^i x(t)$ for each robot j at the field of view of robot i.

- pos_calculator package
  It subscribes "pos$i$", publishes "completedPos$i$" for each robot i and the signal determining whether the transformations for all robot pairs are completed or not ("transformCompleted").
  "completedPos$i$" is the position array consisting of $P_j^i x(t)$ for each robot j after robot frame translations. Differently "pos$i$" topic, it contains $P_j^i x(t)$ for all i, j.

- realization package
  It subscribes "odom$i$", the signal, "completedPos$i$" and publishes the velocity command ("turtle$i$/cmd_vel") for each robot i.

- error_comp package
  It's an optional package to check difference desired $P_j^i x(t)$ and actual $P_j^i x(t)$ calculated by pos_calculator package. It's implemented because it's too difficult to check errors, visually.

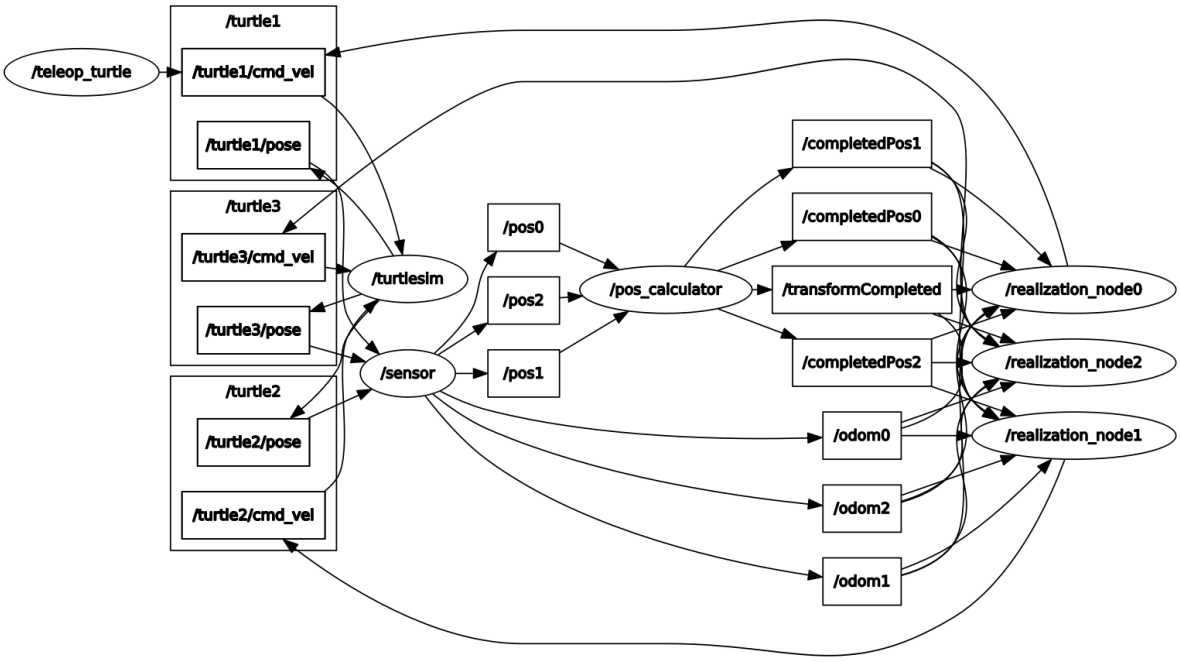


Figure 6: Package Architecture

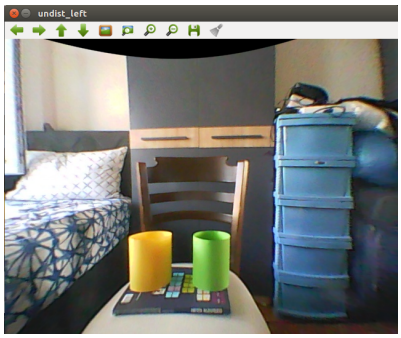
3.4 Object Detection Methods

To be able to use the methods above, the robots are detected and their positions calculated accurately. We tried 4 methods for this purpose.

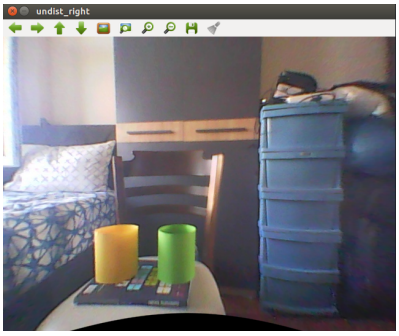
3.4.1 One Color Detection

At the first term project, the objects are detected by using only one color (red object, green object). The code searches the object with these colors and calculates the positions using the disparity method. At the first term, we figure out that when the object are detected and filtered clearly, the disparity method calculates the position accurately. However, there are two main problem. One of them, HSV values of the target object changes depending on light and its angle. Target object may be filtered partially and it causes inaccurate results. The second problem is that there can be another object with the same color. It's a big

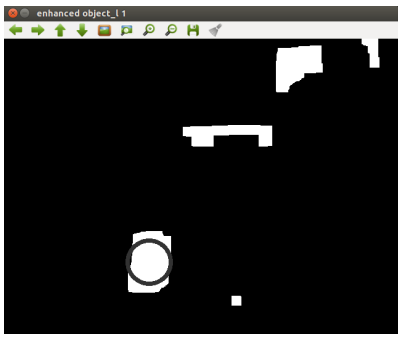
problem. When there is the first problem (inappropriate HSV value), the result has some error, the system considers the part with this HSV values (generally they are same parts for two images came from left and right cameras), however when there is the second problem (another object), the result is completely wrong. In order to solve the second problem, we developed detection algorithm. It searches objects with the HSV value, considers only the biggest part and uses its center of mass to calculate the position. At the initial algorithm, it considered all filtered parts as one part and uses its center of mass. If there is not any bigger object with the same color, the second problem is solved.



(a) undistorted left image

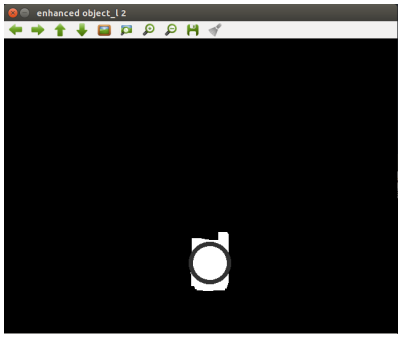


(b) undistorted right image

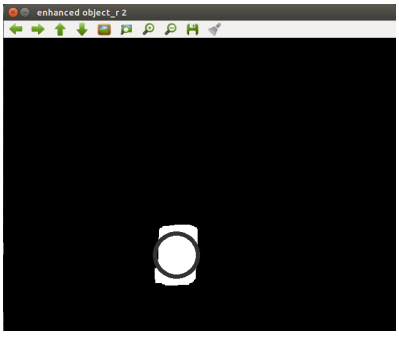


(c) yellow left

(d) yellow right



(e) green left



(f) green right

Figure 7: One Color Detection

When the images at the figures 7a and 7b are used, the target object is yellow and HSV value is defined according to this object however there are some yellow parts at the wardrobe. There are 4-5 parts filtered but the biggest part is the target and the system works properly for yellow and green objects.

3.4.2 Color ID Detection

In order to solve the second problem completely even if there is bigger objects with same color (not solved at one color detection, when we try to detect blue object, it fails due to blue drawer and blue ball), we propose color ID detection method. At this method, there is a color id configuration (top color, bottom color) and each object has one color ID. At this method, the detection algorithm is not affected by other random objects and they are ignored. To be considered, the object must have this color ID.

Firstly, red, green and blue filters are applied and components are extracted. Secondly, the candidate color groups are created according to two constraints. The first one is the difference of two color at horizontal axis must be less than a threshold. The second one is the difference of two color at vertical axis must be more than a threshold. It checks whether they can create a color ID (top-bottom colors) or not. Finally, candidate color groups are checked using target color ID's predefined at the file (figure 8).

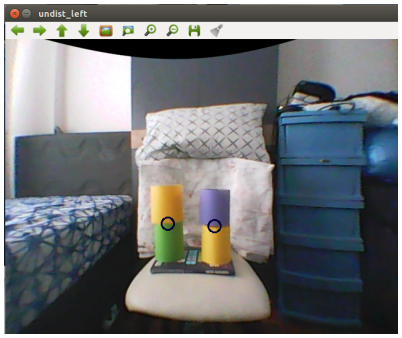
```
//color id: {top color, bottom color}
// 0: yellow, 1:green, 2: purple

const int numrobots = 3;
const int robIDList[3][2] = {{2,0},{1,0},{0,1}};
```

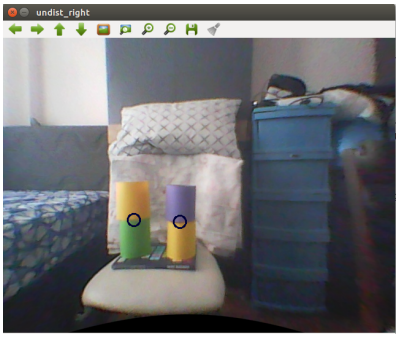
Figure 8: Color ID's Definition

With this method, we can increase HSV ranges and minimize the error due to the first problem because the system ignores unrelated objects. Even if there are green, yellow or purple objects, they are ignored because they can not create a color ID.

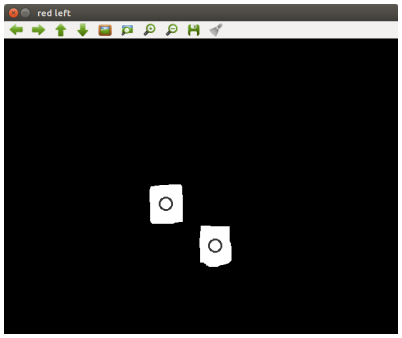
Also, the system gives us the opportunity to detect more objects. We have 5 robots and 9 objects can be defined with 3 colors.



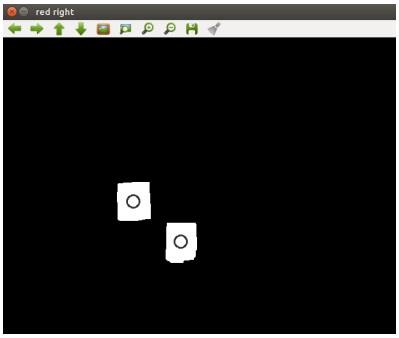
(a) undistorted left image



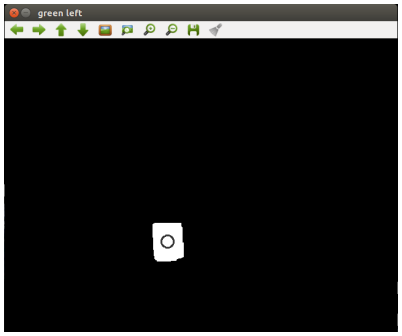
(b) undistorted right image



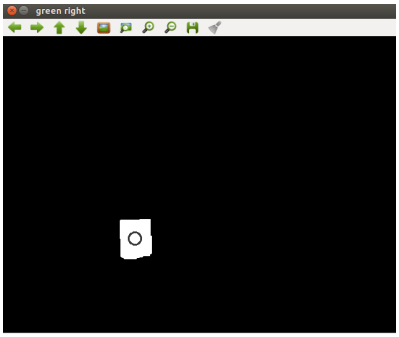
(c) yellow left



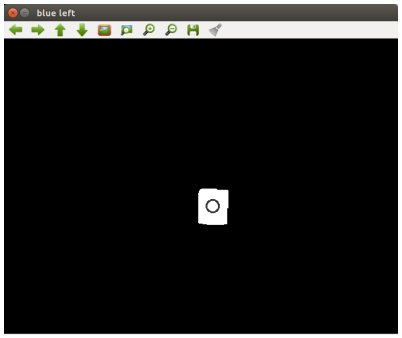
(d) yellow right



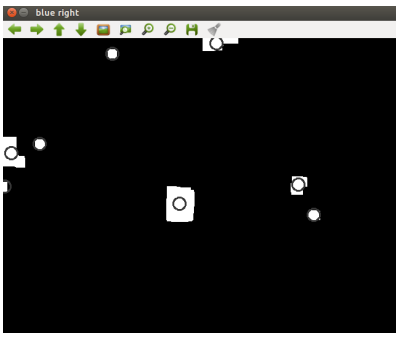
(e) green left



(f) green right



(g) purple left



(h) purple right

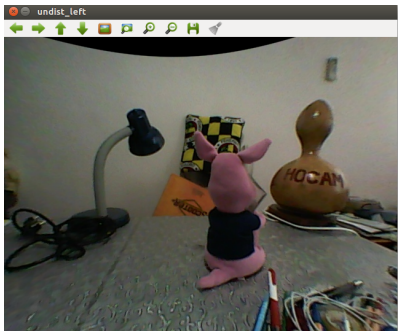
Figure 9: Color ID Detection

At the images at the figures 9a and 9b, the target objects with the predefined color ID's are detected properly.

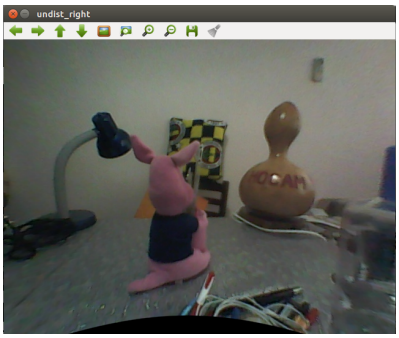
3.4.3 Disparity Matching

At the color ID detection, we solve the second problem (unrelated other objects) but still selection of appropriate HSV values (the first problem) is an important issue. We need to update the values before using the code. We thought we generate disparity map, use segmentation and clustering algorithm and calculate the related objects. To generate clear disparity map, there must be many edges and similar objects must be far enough.

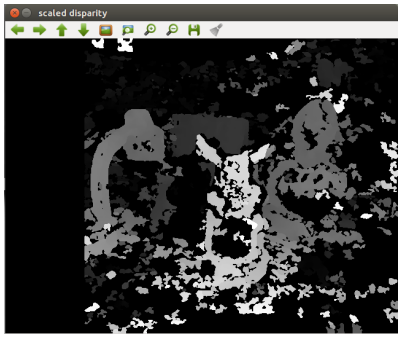
We use openCV disparity matching function and generate disparity map. However, even if there are many objects and edges, it's not clear. Also, the disparity matching algorithm needs computational power. After that, we need to use other segmentation algorithms. Our processor is Raspberry pi3 and it may not have enough computational power. Because of these reasons, we do not use this method.



(a) undistorted left image



(b) undistorted right image

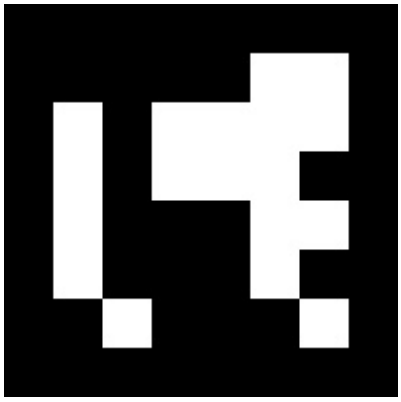


(c) disparity map

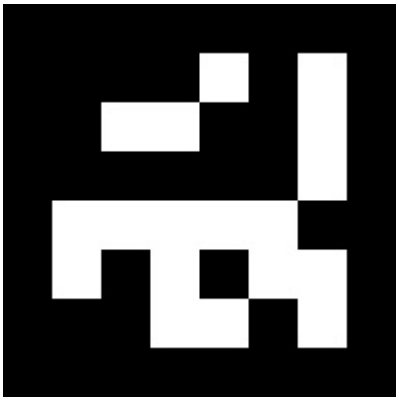
Figure 10: Disparity Matching Method

3.4.4 Aruco Marker Detection

Aruco markers[6] have ID's, they are detected, the position and orientation are calculated easily and accurately. It has many advantages compared to color detection methods. We don't need to find proper HSV and color, one camera is enough to take the position, and we take the orientation of the marker.



(a) ID 1



(b) ID 2

Figure 11: Aruco Markers

We use OpenCV aruco marker functions and they give marker ID, marker position and orientation.

- aruco::getPredefinedDictionary(aruco::DICT_6X6_250)

  This function defines which dictionary will be used. At this dictionary, marker images are defined for marker ID's.

- aruco::drawMarker(dictionary, markerID, 200, markerImage, 1)

  This function takes the dictionary, markerID and creates the marker image.

- aruco::detectMarkers(image, dictionary, corners, marker_IDs)

  This function takes the image, the dictionary and gives the marker corners, marker ID's.

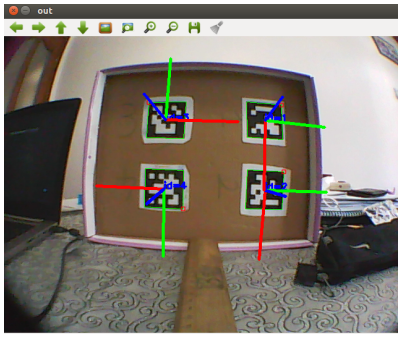
- aruco::drawDetectedMarkers(image, corners, marker_IDs)

  This function takes images, corners, marker ID's and draw the axes related to detected markers.

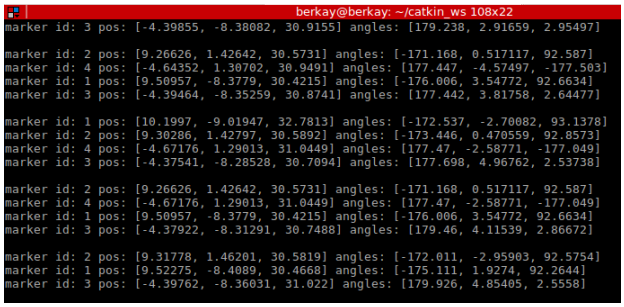
- aruco::estimatePoseSingleMarkers(corners, 5.3, cameraMatrix, distCoeffs, rvecs, tvecs)

  This is the most important function. It takes corners, marker size,camera matrix, distortion coefficients and calculates rotation vectors, position vectors.

  We don't need to make the images undistorted because the function takes camera matrix and distortion coefficients, it considers and fixes only related points (marker corners). As it knows the marker size, one camera is enough.



(a) Markers



(b) ID's, Positions and Angles

Figure 12: Aruco Marker Detection

We create a marker configuration for robots to use markers more effectively. We add aruco markers to each side, robots can be detected, positions and headings of the robots can be calculated using marker positions and orientations if we know on which side the marker is. The heading is related to the angle around y axis. For example, if the aruco is on the front side, the marker angle and the heading are same. If the aruco in on the left side, the difference is 90 degree.

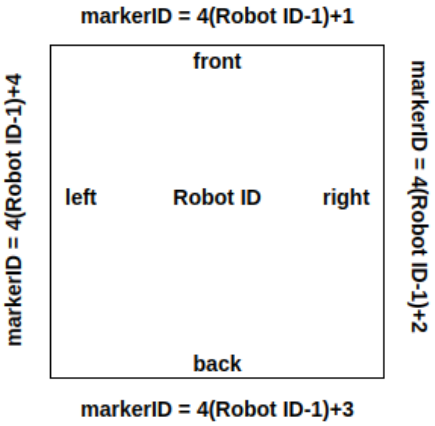
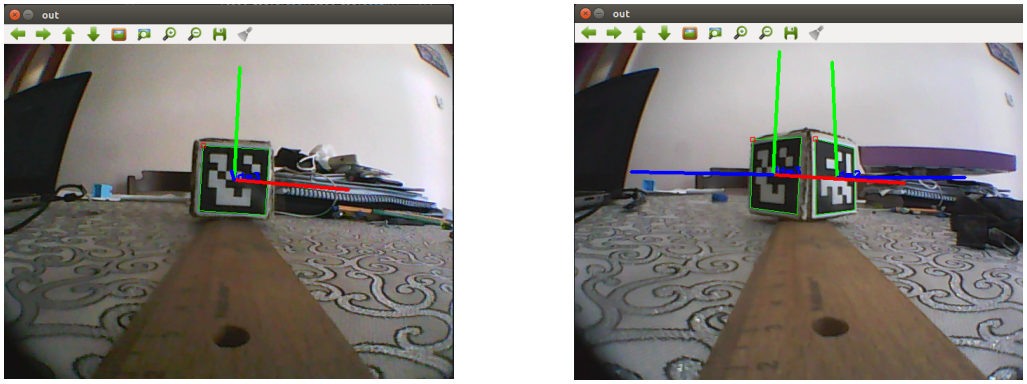


Figure 13: Aruco Marker Configuration

When we use this configuration and one marker is detected, we can understand which robot and which side. Also, using the robot size, aruco marker position is updated to calculate the position of the robot center.

RobotID = (arucoID - 1) / 4 , side = arucoID%4 (1:front, 2:right, 3:back, 4:left)

If the system detects two markers(different sides) with same robot ID, it takes their averages.



(a) One Side Detection (b) Two Side Detection

Figure 14: Cube with Aruco Marker Configuration

3.5 Gazebo Implementation

We would implement on the minik robots however we have to implement on the simulation due to corona virus.

3.5.1 Minik Model

We create differential drive robot with stereo camera systems using URDF description [7] format at gazebo. To add the aruco markers, we use Blender program[8] and create the visual part as DAE file.

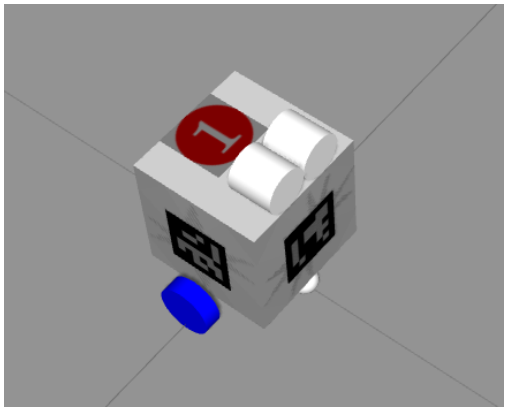


Figure 15: Minik Model

3.5.2 Implementation with Overhead Camera (Complete Information)

Gazebo simulator publishes position data (robot$i$/odom) and realization package takes this data, calculates and publishes velocities.

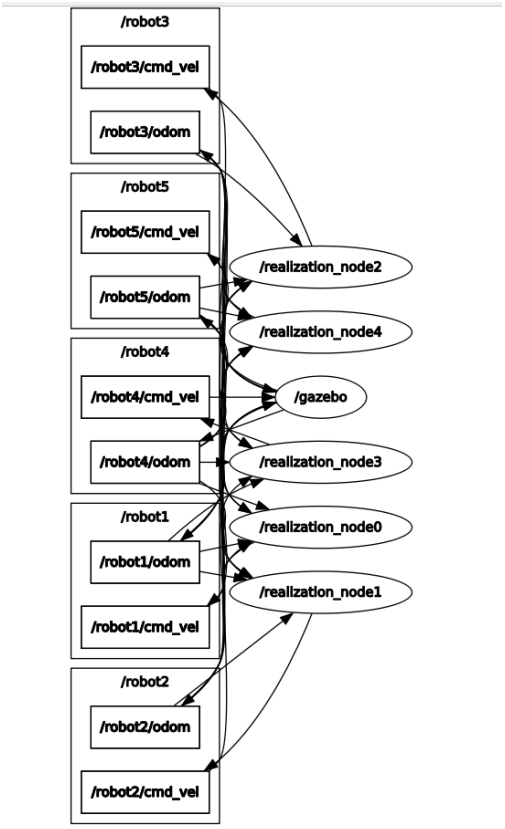


Figure 16: Package Architecture

3.5.3 Implementation with Stereo Cameras (Partial Information)

There are four packages:

- odom_publisher_node

  This packages calculates odometry data with respect to initial robot frame. Gazebo simulator publishes position data (robot$i$/odom) however this data is with respect to world frame. At real cases, robots don't know the positions with respect to world frame. It takes robot$i$/odom topics and publishes odom$_i$.

- detect_aruco_marker

  This package takes the images (robot$i$/multisense_sl/camera/left/image_raw) came from stereo cameras at the gazebo, takes odometry data (odom$_i$) calculates the robot positions with respect to the the initial robot frame for each robot using aruco marker functions and derivations at 3.2 and publishes them (detected_aruco$i$).

  detected_aruco$i$ is the array consisting of robots detected by the robot$i$. This data has robots' ID's, positions and headings.

- pos_calculator

  It's similar to pos_calculator package at 3.3.7. The topic names are different because of different simulator.

  Methods at 3.3.1 and 3.3.2 are implemented.

  This package takes the position data (detected_aruco$i$) came from detect_aruco_marker package, calculate transformations among initial robot frame, calculate each current robot position with re-

spect to each initial robot frame, and publishes them (completedPos$i$) and the completed signal (transformCompleted$i$.

completedPos$i$ is the array consisting of all robots' position with respect to initial robot$i$ frame.

- realization

  It's similar to realization package at 3.3.7. The topic names are different because of different simulator.

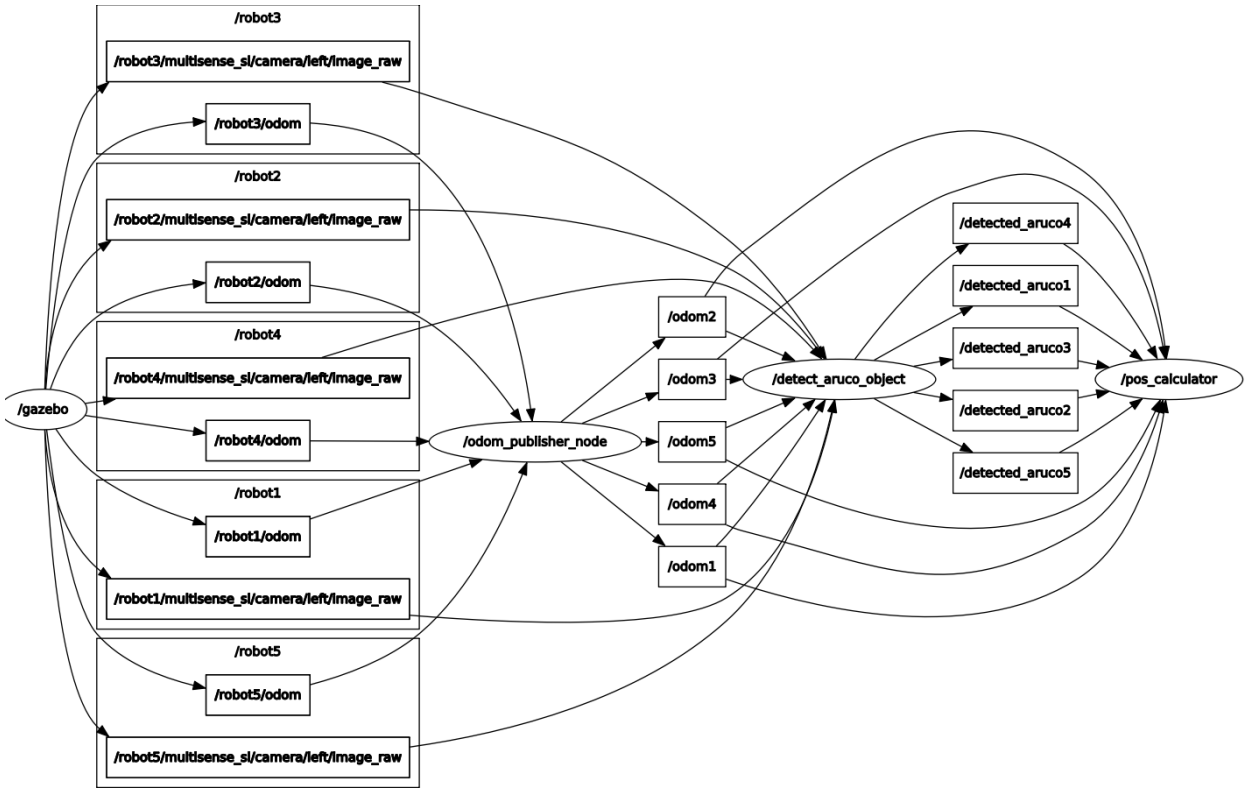


Figure 17: Package Architecture without Realization

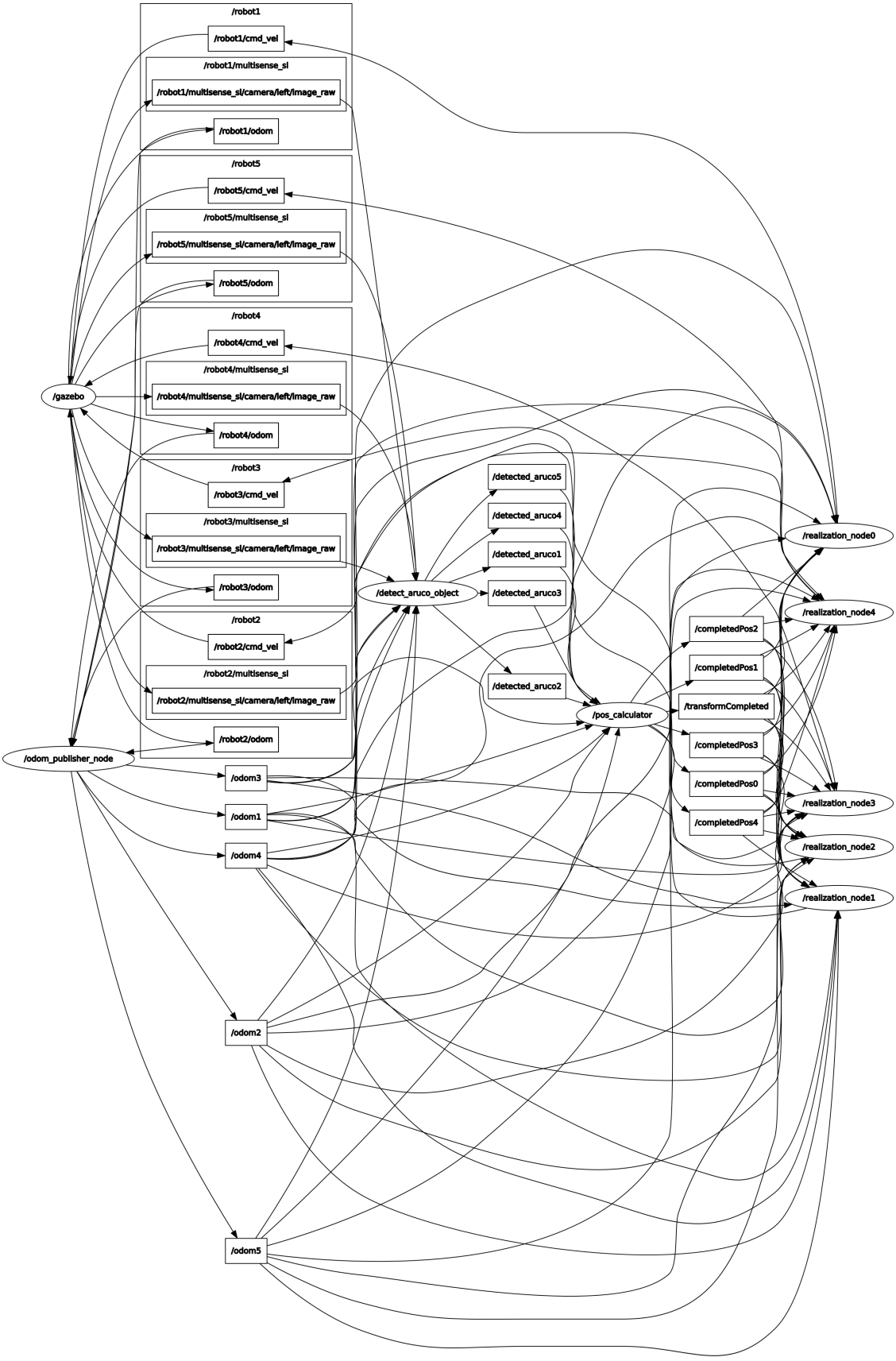


Figure 18: Package Architecture with Realization

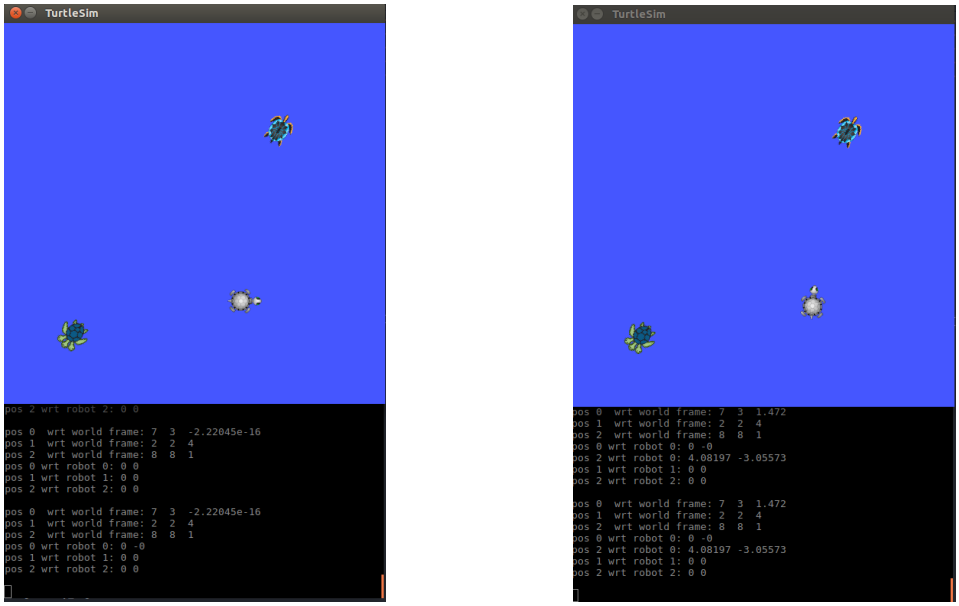
4 Results

4.1 Turtlebot Simulation Results

Three packages(sensor, pos_calculator, realization) are tested separately.

4.1.1 Sensor Package Results

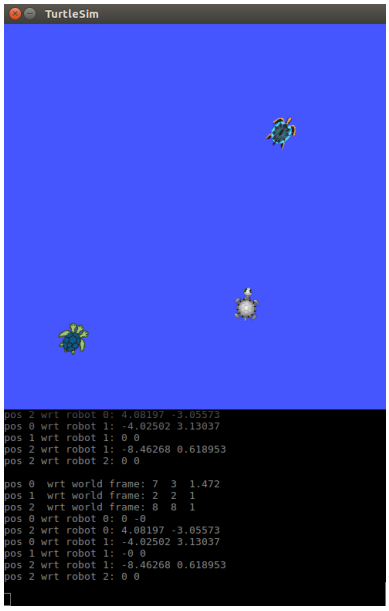
Firstly, the sensor package results are checked. The simulation workspace is 10x10.



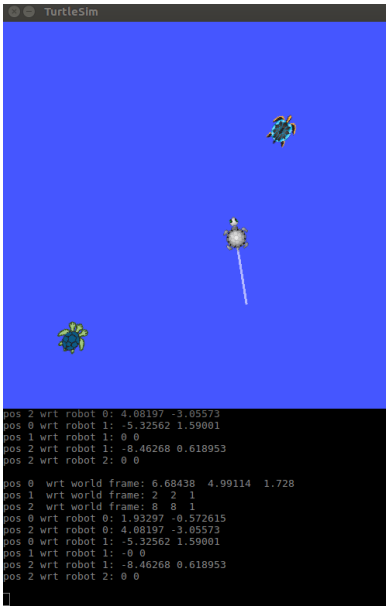
(a) Configuration 1 (b) Configuration 2

Figure 19: Test Configurations(1-2) for Sensor Package

At the configuration 1, robots can not observe other robots and there is not the position data from the stereo camera system. There are only odometry data ($P_i^i x(t)$). Moreover, all odometry data is 0 because the robots have not moved yet.
At the configuration 2, only robot0 can observe robot2 and the sensor package publishes $P_2^0 x(t)$. There are only odometry data.



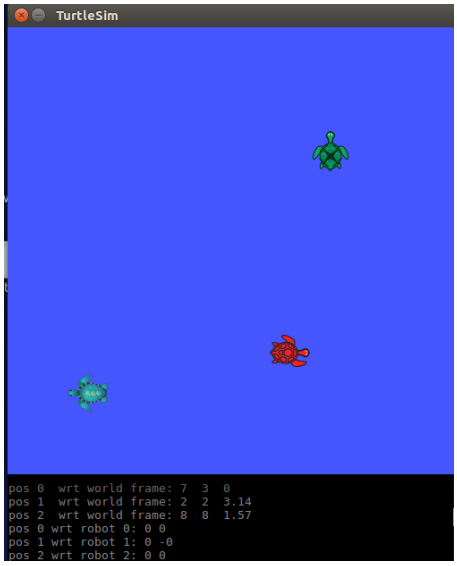
(a) Configuration 3



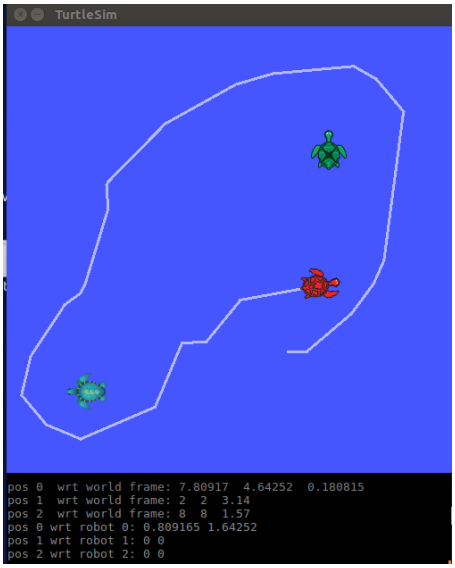
(b) Configuration 4

Figure 20: Test Configurations(3-4) for Sensor Package

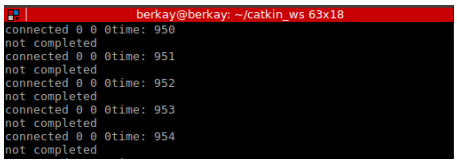
4.1.2 Pos_calculator Package Results



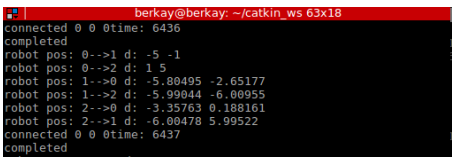
(a) initial configuration and sensor results



(b) final configuration and sensor results



(c) pos_calculation results



(d) pos_calculation results

Figure 21: Test Case for Pos_calculator Package without Noise

("robot pos: i->j d:" means position of robot j with respect to initial frame of robot i)
At the initial and final configurations, robots can not observe each others and sensor package publishes only odometry data. However, pos_calculator packages give different results. At the initial configurations, it doesn't give anything but at the final configuration, it gives $P_i^i x(t)$ for each i, j. Because, the robot 0 moves manually to make robot 1 and robot2 observe robot 0 at two different points in order to be able to method 1. After these observations, robot transformations are completed using method 1 and each robot position can be calculated even if it is not at the field of view.

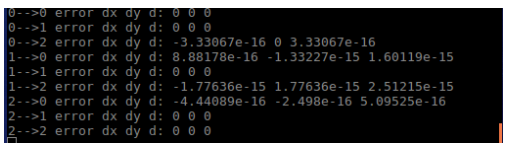


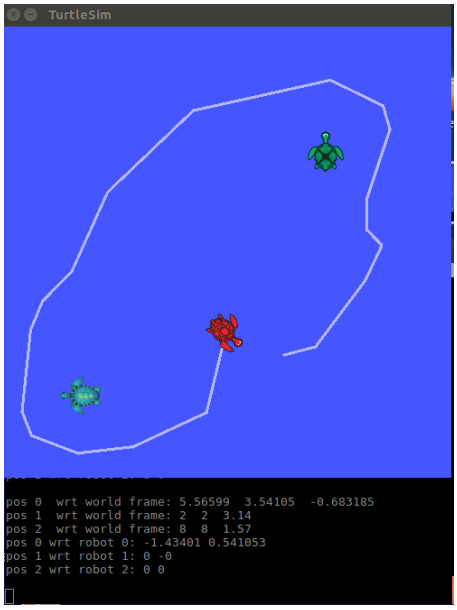
Figure 22: Error Results

("i->j error dx dy d:" means error at position of robot j with respect to initial frame of robot i for

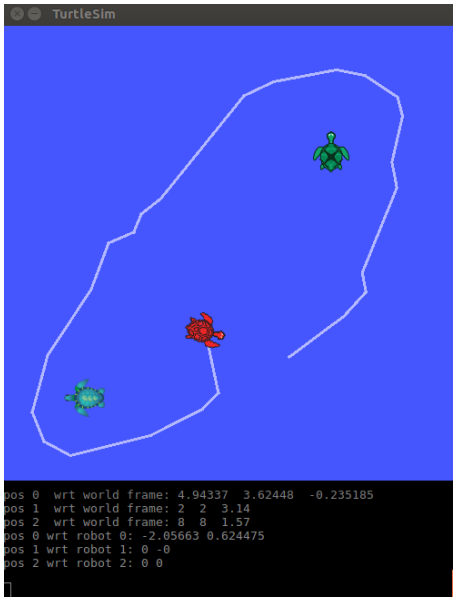
x axis, y axis and distance)

The pos_calculator package works properly and errors are almost zero when the sensor is noiseless. The random noise is added to the position data for the stereo camera system. The system is tested for the same initial configuration, similar path for robot O and similar final configurations. At the left case, the random noise between -0.1 and 0.1, at the right case, it is between -0.5 and 0.5.

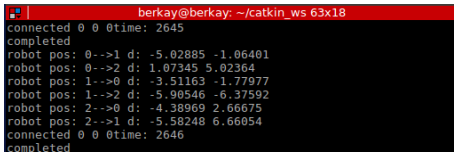
(The workspace is 10x10 and the robot diameter is about 0.8)



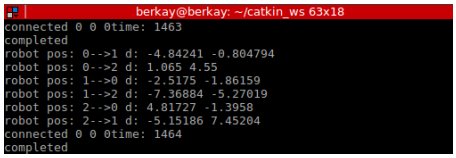
(a) final configuration and sensor results



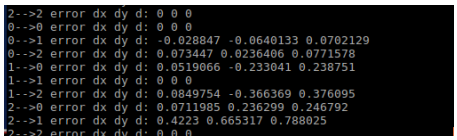
(b) final configuration and sensor results



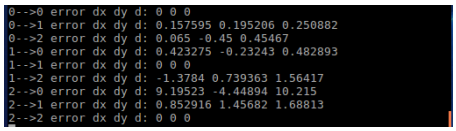
(c) pos_calculation results



(d) pos_calculation results



(e) error results for random noise with 0.1



(f) error results for random noise with 0.5

Figure 23: Test Cases for Pos_calculator Package with Noise 0.1 and 0.5

For the system with 0.1 random noise, the system works properly, errors are less than 1 and it's compensated. However, the system with 0.5 random noise, some calculations have too much errors, especially $P_0^2 x(t)$ whereas other results have errors about 1.

4.1.3 Realization Package Results

For simulation,the radii of the robots are set as 0.5 meters. The test case with 3 robots is shown below. The adjacency matrix A is as follows: $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ The adjacency threshold is $\rho = 2.$ and k is 3.

Initial configuration is same as previous cases, each robot pair is not close and the goal is to be closer than 2. The sensor is noiseless.

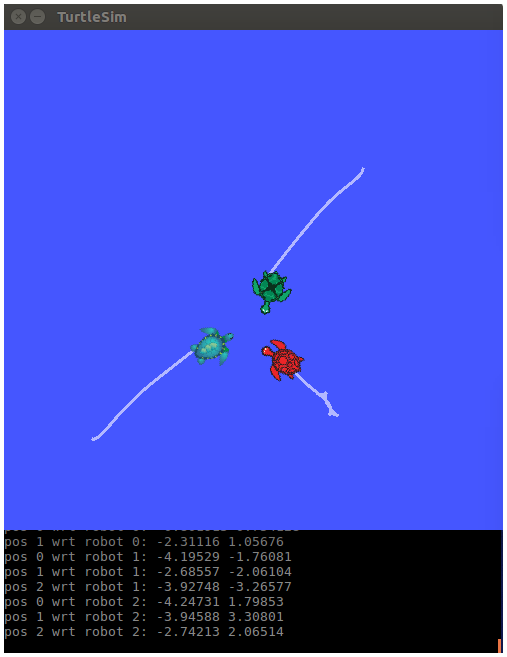
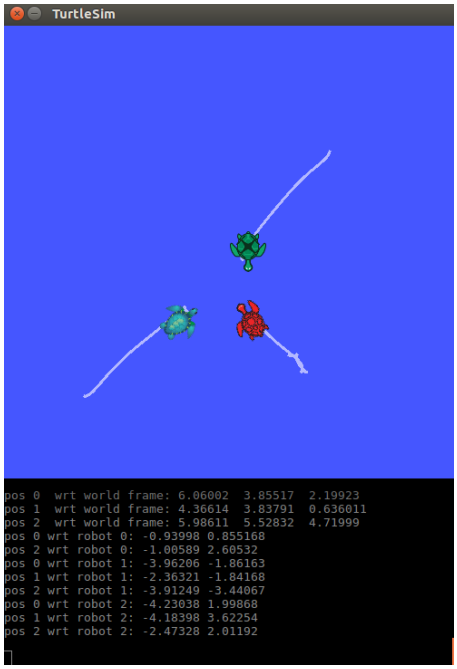
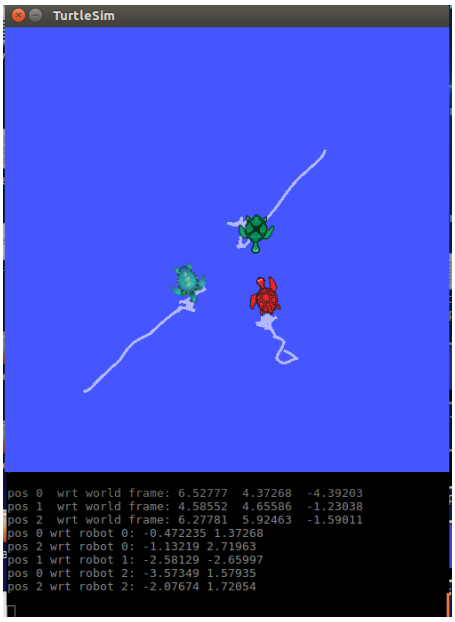


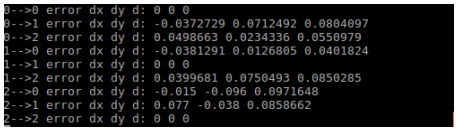
Figure 24: Final Configuration without Noise



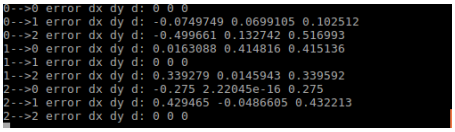
(a) final configuration for random noise with 0.1



(b) final configuration for random noise with 0.5

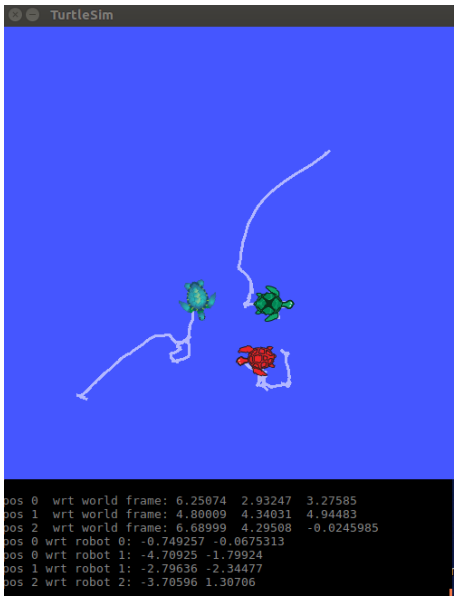


(c) error results for random noise with 0.1

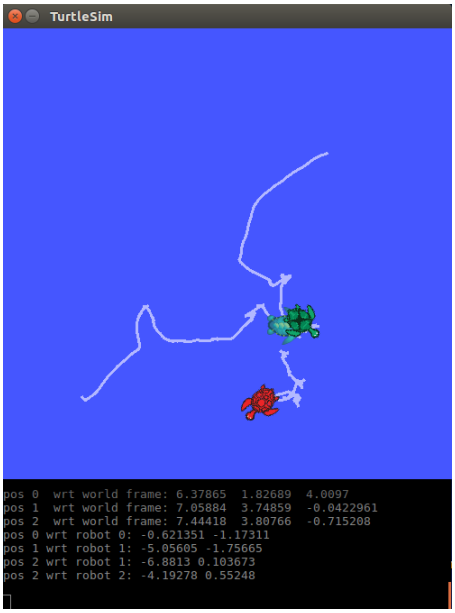


(d) error results for random noise with 0.5

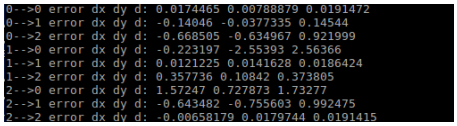
Figure 25: Test Cases for Realization Package with Noise 0.1 and 0.5



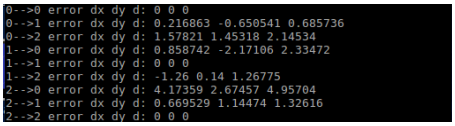
(a) final configuration for random noise with 1



(b) final configuration for random noise with 2



(c) error results for random noise with 1



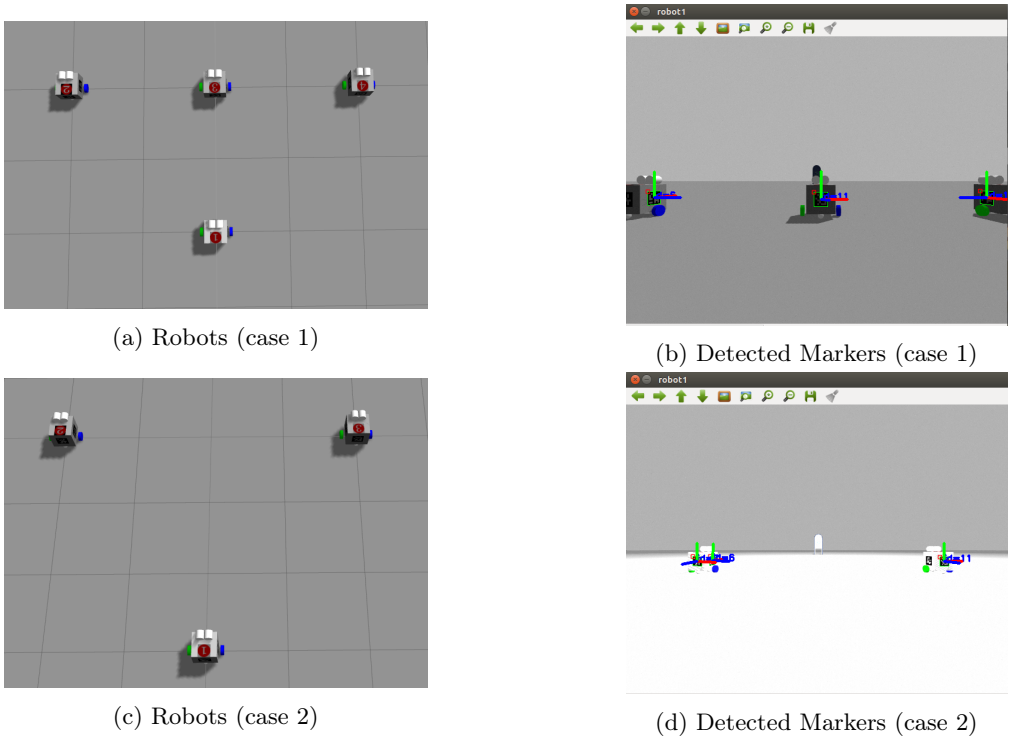
(d) error results for random noise with 2

Figure 26: Test Cases for Realization Package with Noise 1 and 2

For the random noise with 0.1, the system works properly and smoothly. When the noise increases, the smoothness of path decreases. For the noises with 0.5 and 1, the error at positions are not too much and system works. However, for the random noise with 2, the system fails and robots crash.

4.2 Gazebo Simulation Results

4.2.1 detect_aruco_object Package Results



(a) Robots (case 1)

(b) Detected Markers (case 1)

(c) Robots (case 2)

(d) Detected Markers (case 2)

Figure 27: Aruco Marker Detection at Gazebo

real pos(x,y)	calculated pos(x,y)
2, -2	2.05,-2.04
2, 0	2.04, 0.04
2, 2	2.05, 2.05
3, -2	3.08, -2.05
3, 2	3.15, 2.12

4.2.2 Realization Results

Four goal topology is tested for the same initial topology using two realization methods (overhead camera and stereo camera). N is set as 5, K as 6, ρ_{ij} as 0.5 and threshold as 3.

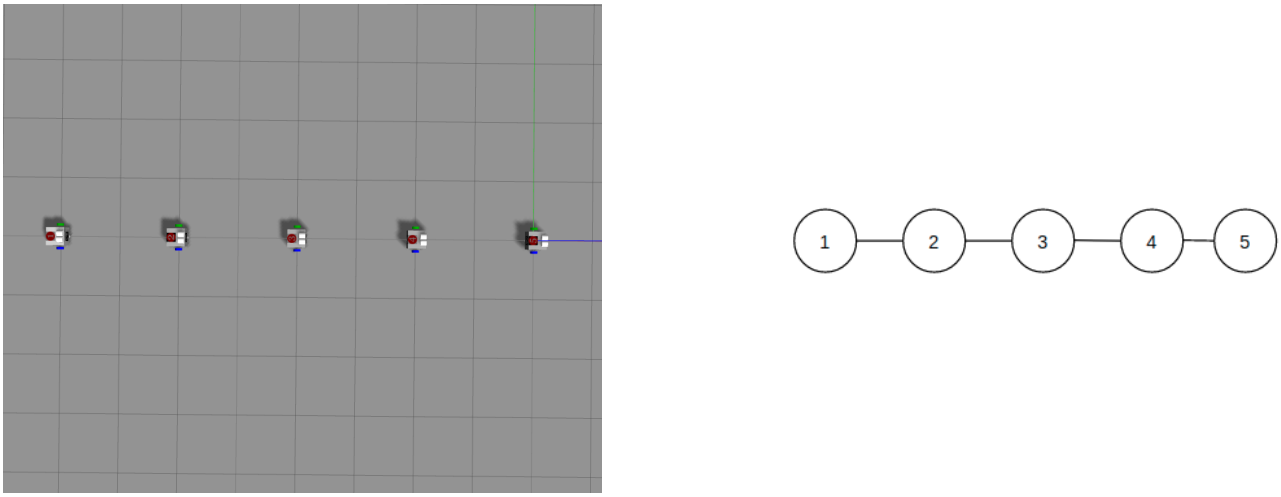


Figure 28: Initial Topology

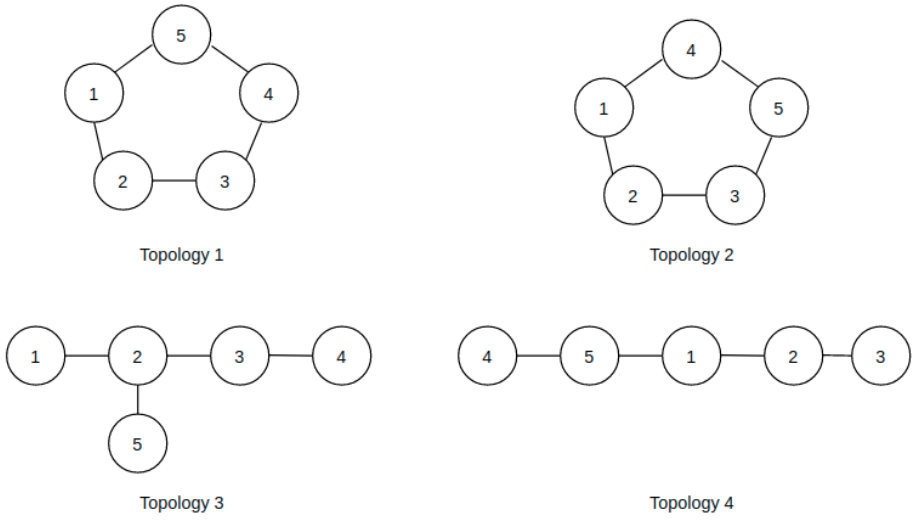
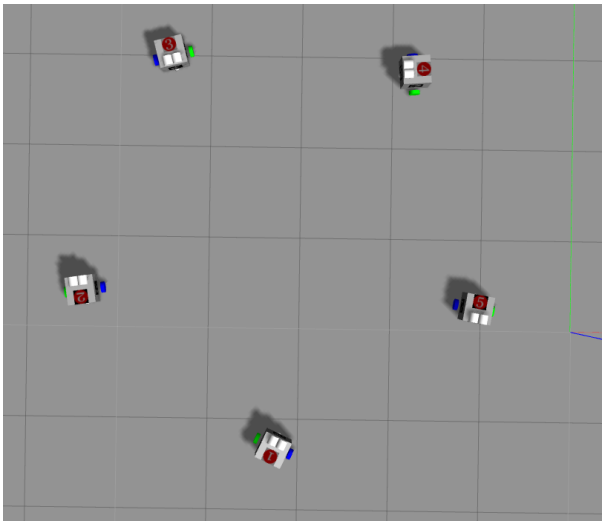
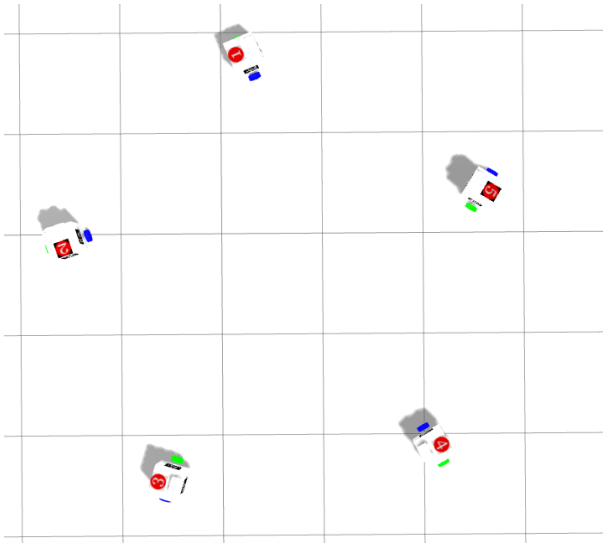


Figure 29: Goal Topologies

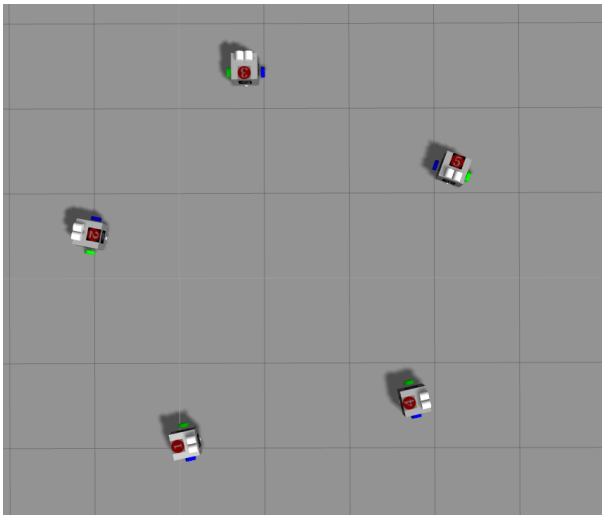


(a) with overhead camera

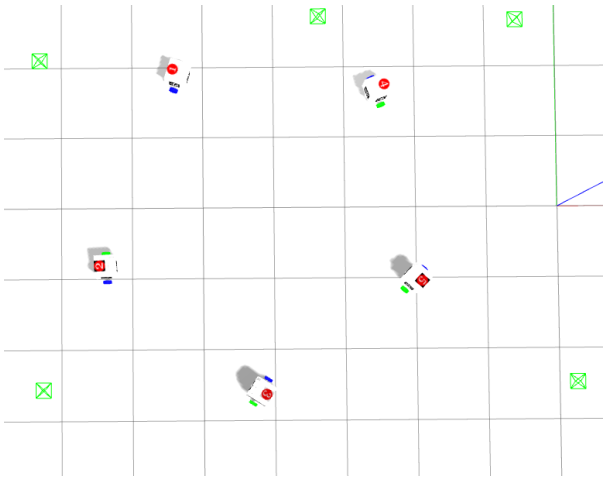


(b) with stereo camera

Figure 30: Goal Topology 1

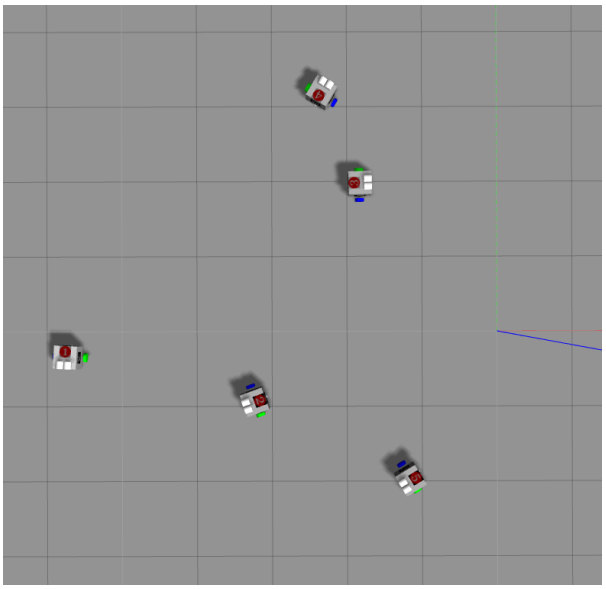


(a) with overhead camera

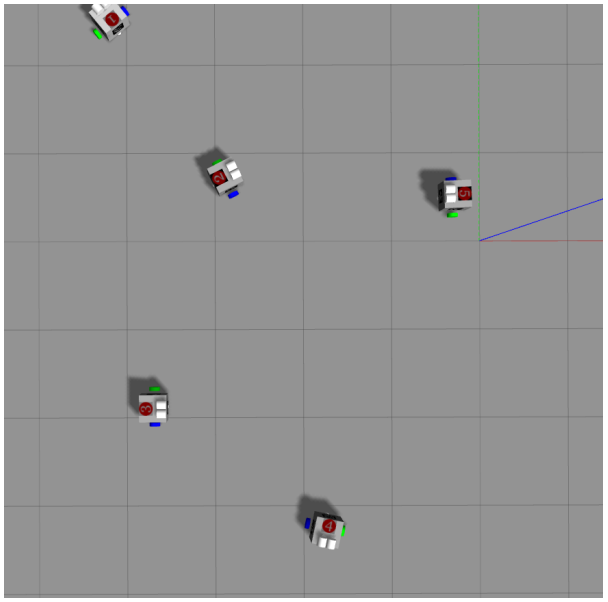


(b) with stereo camera

Figure 31: Goal Topology 2

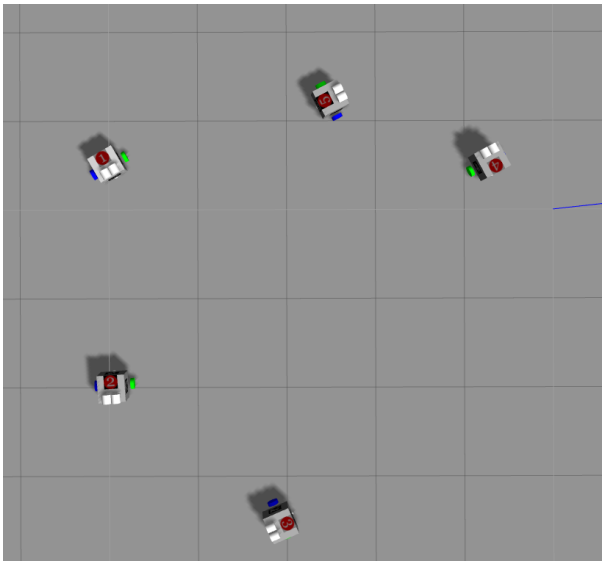


(a) with overhead camera (fail)

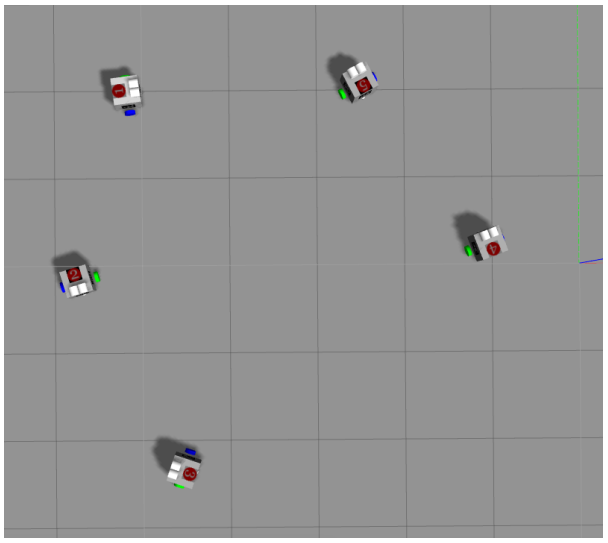


(b) with stereo camera

Figure 32: Goal Topology 3



(a) with overhead camera



(b) with stereo camera

Figure 33: Goal Topology 4

These four goal topologies are realized with stereo camera system whereas three of them are realized with overhead camera system.

5 Conclusion

5.1 Turtlesim

We tested the system with random noises and it doesn't depend on the distance between observed and observed robots. At real scenarios, the error of stereo camera system depends on the distance, it's not constant values such as 0.1 and 0.5. The reason that the case at figure 23b fails and has too much error but the case at figure 25d works and has small error is that both of them have same error (random noise with 0.5) but error rates are different. The distance at the first case about 0.5 and error rate is about 100% whereas at the second case, the error rate is not too much because of the distance.Also,the cases with more than 3 robots and other robot topologies must be tried.

Moreover, if we can find and adapt new methods for the transformations, the system can be more robust.

5.2 Gazebo

When we tested 4 topologies many times, the goal topologies were realized, there were not any problem. The system on gazebo works properly. Images came from the stereo camera has noise but odometry data doesn't have noise. To test how robust the system is, the noise can be added to odometry data. Also, the inertia properties of the robots can be more proper.

5.3 Social, Environmental and Economical Impact

Over recent years, the importance of multi-robot system has increased. As the number of robots is increasing, social life is affected much more from robot technology, especially after increase the computational power and topics such as deep learning and artificial intelligence. These multi-robot system can do easily and automatically any task which humans or one robot cannot make or make hard. Their usage areas can be search and rescue, health organizations, emergency situations, transportation and autonomous factory systems.

5.4 Cost Analysis

There were many hardware of the projects at the lab and we bought the necessary equipment such as lipo batteries and fish-eye lens at the first term. There is only labor cost which can be calculated as 180 hours * 50 TL/hour = 9000 TL

5.5 Standards

The implementation, design and development of this project apply to available engineering standards, including IEEE, IET, EU and Turkish standards along with the engineering code of conduct.

References

[1] Spong, M.W., S. Hutchinson, M. Vidyasagar, Robot Modelling and Control, John Wiley & Sons Inc, 2006

[2] http://www.isl.ee.boun.edu.tr/courses/ee451/lectures/mc.pdf

[3] Şenel, Deniz, H. Işıl Bozma, and Ferit Öztürk. "Multi-Robot Realization Based on Goal Adjacency Constraints." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.

[4] Şenel, Deniz, H. Işıl Bozma, and Ferit Öztürk. "Finding optimal isomorphic goal adjacency." Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016.

[5] https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf

[6] Docs.opencv.org. 2020. Opencv: Detection Of Aruco Markers. [online] Available at: <https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html> [Accessed 11 July 2020].

[7] Gazebosim.org. 2020. Gazebo : Tutorial : URDF In Gazebo. [online] Available at: <http://gazebosim.org/tutorials?tut=ros_urdf> [Accessed 11 July 2020].

[8] Foundation, B., 2020. Blender.Org - Home Of The Blender Project - Free And Open 3D Creation Software. [online] blender.org. Available at: <https://www.blender.org/> [Accessed 11 July 2020].