Final Design Review (FDR) report

# Pseudo GPS for Romi
# Final Design Review

December 9, 2024

For Charlie Refvem


Team

Emmanuel Baez – embaez@calpoly.edu

Gabriel Coria – gacoria@calpoly.edu

Owen Guinane – oguinane@calpoly.edu

Conor Schott – cjschott@calpoly.edu

College of Engineering

California Polytechnic State University, San Luis Obispo

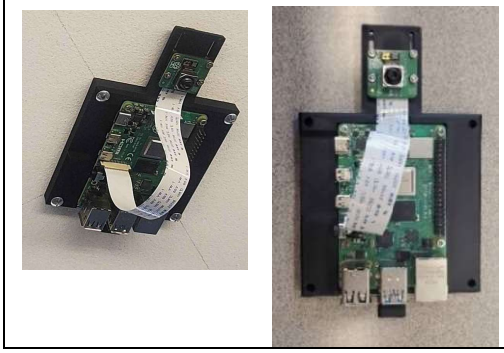

2024

# Statement of Disclaimer

Since this project is a result of a class assignment, it has been graded and accepted as a fulfillment of the course requirements. Acceptance does not imply technical accuracy or reliability. Any use of information in this report is at risk to the user. These risks may include catastrophic failure of the device or infringement of patent or copyright laws. California Polytechnic State University at San Luis Obispo and its staff cannot be held liable for any use or misuse of the project.

# Four Panel Chart

**Table 1.** Pseudo-GPS for Romi Bots

<table>
<tr>
<td>Project Overview<br><br>
<ul>
<li>Problem Statement: Currently, the Romi robots in the Cal Poly mechatronics lab lack the capability to obtain absolute orientation or location from onboard sensors. To support the development of advanced robotic algorithms, ME 405 Instructor, Charlie Refvem and Mechatronics students require a system that can deliver precise, real-time location data for multiple Romi robots. This project aims not only to improve tracking accuracy but also to enhance communication standards across Romis, focusing on geolocation precision, range, and latency.</li>
<li>Team Members: Emmanuel Baez, Gabriel Coria, Owen Guinane, Conor Schott</li>
<li>Sponsor and Stakeholders: Charlie Refvem (Instructor, ME 405)</li>
</ul>
</td>
<td>Design Description<br><br>
<ul>
<li>The system tracks Romi robots within a lab environment using a Raspberry Pi 4 equipped with a Pi camera module 3 and ArUco markers.</li>
<li>Key Components:
<ul>
<li>Markers: Numbered ArUco markers mounted on custom 3D-printed stands.</li>
<li>Camera Module: A Pi camera module 3 is configured for accurate tracking</li>
<li>Origin Marker: A designated ArUco marker serves as the origin for positional calculations (we had the origin as “1” ArUco marker for our program)</li>
<li>Tracking Software: Python programs compatible with the Raspberry Pi hardware were developed to calculate marker positions and orientations, and to support future features.</li>
</ul>
</li>
</ul>
</td>
</tr>
<tr>
<td>Design Image<br><br>

</td>
<td>Design Verification<br><br>
<ul>
<li>Target Specifications:
<ul>
<li>Positional Accuracy: ±5mm</li>
<li>Orientation Accuracy: ±1 degrees</li>
<li>Latency: 10HZ</li>
</ul>
</li>
<li>Key Results:
<ul>
<li>Positional Accuracy: ± 7.23mm</li>
<li>Orientation Accuracy: N/A</li>
<li>Latency: N/A</li>
</ul>
</li>
</ul>
</td>
</tr>
</table>

# Overview

The Romi robots in the Cal Poly Mechatronics lab cannot currently obtain absolute orientation or location from onboard sensors. To support the development of more advanced algorithms, ME 405 instructor Charlie Refvem and his students need a system that provides precise, real-time location data for multiple Romi robots. This solution aims to improve not only tracking but also communication standards across robots by ensuring high levels of table location accuracy, precision, range, and low latency.

# Concept Description

The purpose of this project is to track Romi robots within a lab environment using a Raspberry Pi equipped with a camera module and ArUco markers. Each Romi is outfitted with a unique marker attached to a custom 3D-printed stand, while additional markers can be placed on obstacles or their corners to provide positional context relative to the robots. The system tracks Romi's positions and identifies obstacles within the environment to help support autonomous navigation.

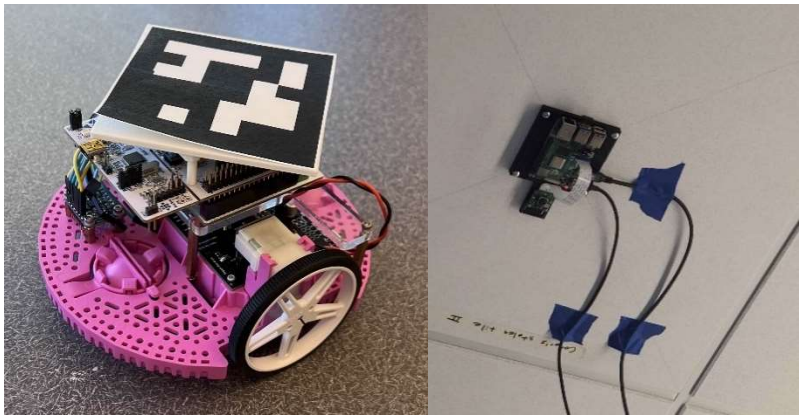


Figure 1. The left-hand image showcases a Romi robot equipped with an ArUco marker securely mounted on top using a custom-designed 3D-printed attachment. The right-hand image features the ceiling-mounted setup, which integrates a Pi camera and a Raspberry Pi for overhead monitoring and processing.

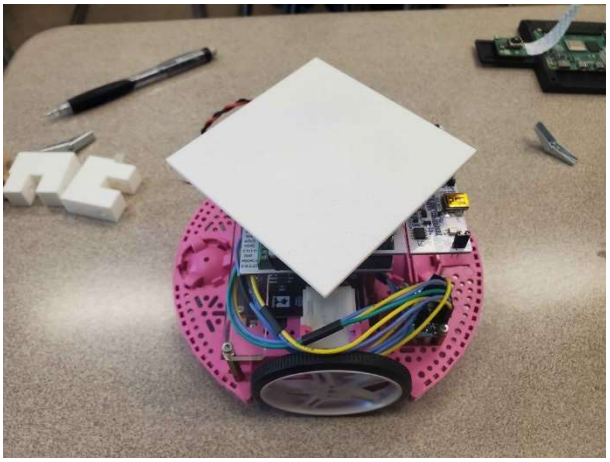


Figure 2. Picture of Romi before placing ArUco Marker on mount

Markers are printed on standard printer paper, cut out, and mounted on 3D-printed stands that attach securely to the Romis or obstacles. A designated ArUco marker serves as the origin, with coordinates for other markers calculated relative to it. The camera's configuration, including a fixed height and field of view, ensures accurate tracking when markers are properly aligned, and environmental lighting is sufficient.

Three programs support the system's functionality:

- **Marker Generator**: Creates ArUco marker images for printing.
- **Position Tracking Program**: Calculates marker positions at a fixed height relative to the origin.
- **Orientation and Position Tracking Program (Uncalibrated)**: A prototype program for future use, incorporating orientation tracking features not yet fully implemented or tested.

In its current form, the system processes positional data locally. Future iterations aim to enhance functionality by enabling data transmission to an ESP32 microcontroller over Wi-Fi, facilitating integration with additional robotic systems or control platforms.

# Implementation

To manufacture our system, we procured parts from a variety of sources. The ceiling tile on which the pi is mounted was procured from the Wood Shop (Facilities, building 70) and is specific to labs within Engineering IV. The M2.5 screws and nuts, M2 screws and nuts, and heat-set M2.5 inserts were purchased from Amazon.com. The 1/8" Mushroom-head toggle bolts used to secure the main housing to the ceiling tile were provided by Cal Poly maintenance following a review of our design. The Raspberry Pi 4, Pi camera modules, MicroSD cards, and cables were procured from Adafruit.com. The main housing and camera spacers were printed out of Overture

PLA plastic using a BambuLab A1 Mini 3D printer. These parts were printed using the default settings for generic PLA on the printer, using a 15% infill and normal supports.

The system was assembled by first inserting the M2.5 heat-set inserts into the four inner holes of the main housing using a soldering iron.

Figure 3. Main Housing for Raspberry Pi 4 getting M2.5 heat-set inserts

Then, the Raspberry Pi 4 is secured to the main housing using four M2.5 hex-head screws. Four M2 screws are then used to hold the camera spacer from the bottom between the two slots on the main housing. After this step, the Raspberry Pi camera module is placed on top of the four screws and secured to the housing using four M2 nuts. The camera module is then connected to the pi using a ribbon cable. The housing assembly is then secured to the ceiling tile using four 1/8” Mushroom-head toggle bolts. The final assembly is then used to replace a ceiling tile within the lab, where power and display cables are connected to the Raspberry Pi.

To set up the software, several Python libraries must be installed and configured in the Thonny Python environment. These libraries are critical for running the provided code on the Raspberry Pi. The required libraries and their primary functions are as follows:

- cv2: OpenCV, used for image processing and computer vision tasks like ArUco marker detection.
- numpy: For handling matrix and array-based computations.
- picamera2: To interface with the Raspberry Pi camera.
- time: For managing delays or intervals, like timing for marker detection.
- math: For handling mathematical operations such as angles or trigonometry.
- os, sys: For system operations, managing file paths, and handling the environment.

• threading: If you’re working with concurrent tasks.

• argparse: For parsing command-line arguments if necessary.

• aruco: For specific functions related to ArUco markers.

After installing the necessary libraries, upload the code to the Raspberry Pi for preliminary testing. To test the camera functionality, open Thonny or Visual Studio Code and run the Python script for the camera. Ensure the camera feed is clear; if it's blurry, adjust the camera settings and calibrations.

Next, to test the ArUco marker detection, run the marker generation script to create the ArUco markers. Print them, place them within the camera’s field of view, and run the marker pose estimation script to verify that all markers are detected, and their positions are accurate. If markers aren’t detected, check the camera alignment, marker visibility, and lighting conditions.

It’s important to ensure the "origin" ArUco marker is in the camera's field of view for continuous monitoring; without it, the program will not display a video feed. To view the video feed, a monitor must be connected to the Raspberry Pi. The terminal on the Pi will display either position or position and orientation data, depending on the program being used. The calibrated position program will output only positions, while the uncalibrated position and orientation program will output both position and orientation data.

## Design Verification

To evaluate our final prototype, we conducted a series of tests in the Mechatronics Lab. As of the time of writing, the rate of transfer tests and fitment and weight tests have not been conducted due to ongoing issues with flashing MicroPython to the ESP32 modules for networking. In addition, the orientation tests were not conducted due to time constraints. Our test results are summarized in Table 2 below.

Table 2: Tabulated results from our testing.

| Test | Target | Result | Pass/Fail |
|---|---|---|---|
| Tracking Resolution | ±5mm | ±7.23mm | FAIL |
| Simultaneous Detection | 6 | 6 | PASS |
| Ceiling Test | 3 hours | 3 hours | PASS |
| Fitment & Weight | <2 lbs. | N/A | FAIL |
| Transfer Rate | 10 Hz | N/A | FAIL |
| Orientation Accuracy | ± 1° | N/A | FAIL |

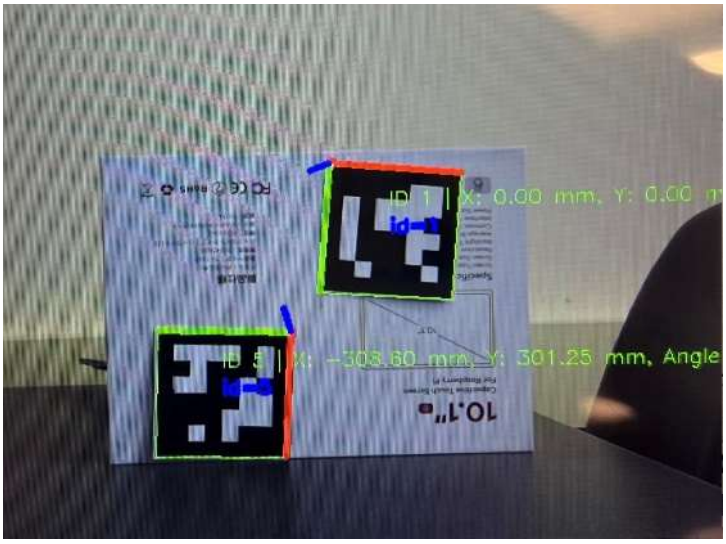



Figure 5. Picture from a testing day showing an origin ArUco marker and another showing its coordinate relative to the origin

One thing we learned from our testing is that the grid we had created to measure the distances was off. The squares from the grid would increase by 1mm after every 5 grid spaces. So, we used the blank side of the grid sheet, because the ArUco markers do well on white background surfaces. To determine our simultaneous marker detection we placed 6 markers, including the origin marker, in the field of view of our system and on the white background. What we were looking for was that each marker was being detected and that the system was able to give positional data for what it thought to be the center of the marker.

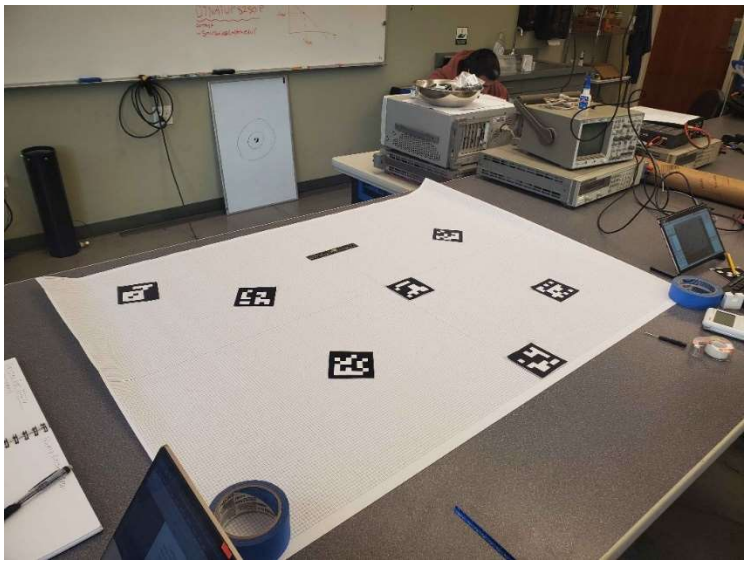


Figure 6. Grid used with multiple ArUco markers placed on top for testing in the mechatronics room

Seeing that the system was easily able to see the Aruco markers, we started to test the accuracy of the positional data the system was giving us. We did this by placing the ArUco markers in various places in the field of view of our system and manually measured the center of one ArUco marker to the center of our origin ArUco marker. We saw that having the markers just set on the white background with no various height changes was giving okay results, we wanted to see how reliable the system would be at various heights. So, using the ArUco marker stands we designed for the Romi we tested the accuracy of our system at the height of the Romi with our stand. These results fluctuated a bit more than our baseline desk height, so we did one more test of placing the markers at various heights. We were able to do this by using objects found in the lab and just placing a marker on them. The results of these tests are summarized in Table 3 below.

Table 3. Tracking Resolution Test Results

| Tracking Resolution Test Result Summary | | |
|---|---|---|
| | Avg. Resolution [mm] | Std. Deviation [mm] |
| Constant Height Test (Z = 0mm) | 4.70 | 3.47 |
| Constant Height Test (Z = 105 mm) | 10.70 | 3.77 |
| Variable Height Test | 6.30 | 4.83 |
| Grid Map Uncertainty [mm] | 1 | 1 |
| Overall Results | 7.23 | 7.04 |

Throughout our testing, we noticed that height played a key factor in determining how accurate our results were. Our system performed best under the constant height condition at z = 0mm, with an average tracking resolution of 4.70 mm with a standard deviation of 3.47 mm. While on average it performs within the target, our uncertainty cannot guarantee that accuracy. Our system performed the worst under the z=105 mm constant height condition, with a tracking resolution of 10.7 mm with a standard deviation of 3.77 mm. The various height test results were in the middle, with an accuracy of 6.30 mm and a standard deviation of 4.83mm. Overall, the system had a tracking accuracy of 7.23 mm with a total uncertainty of 7.04 mm, which is below our specification target for our system.

A potential solution for these inconsistencies is to have a standardized height at which the markers should be detected for the best results. If we have a constant vertical distance from our system where markers should be, then we could calibrate the system to account for that height better.

While we were able to complete tracking resolution, simultaneous detection, and ceiling tests. The fitment and weight, transfer rate, and orientation accuracy tests were unable to be completed. In the case of the fitment and weight tests, ongoing issues with flashing MicroPython to the ESP32s which the system uses to transfer data to the Romi robots were not resolved in time for testing. In addition, orientation tests were not performed due to time constraints and lower priority in comparison to tracking resolution.

# Conclusion

In conclusion, our system performs below what we expected but shows promising results. This project serves as a proof of concept for a system that could be capable of having some way to do local tracking. We were able to make sure that multiple markers could be detected at any time of use for the system. As we were not able to reach the targeted goal of having a tracking resolution of ± 5mm we were able to see our system was off on average by 7.23mm. Upon further refinement of the project, one could easily pick up where we took off. One of the issues we found that affected the tracking resolution was the position of the ArUco marker on the z-axis

(how close to the camera they were). The way the program has it is that it is calibrated by having all the markers at the same height as being at least 6.25 ft away from the system. One could set a baseline height that could be used during actual runs. Our sponsor Charlie wants this height to be relatively fluid so that the students can design their bracket that allows their wiring and allows a marker to be held.

For the tests we were unable to perform, this would be because of time constraints. To get the system for that kind of testing one would need to get two ESP32s. The ESP32 is a microcontroller that is C++ base. So, one could code on it using the Arduino IDE, but then the coding would be done in C++ and not Micropython. Since the current system uses Thonny and runs with MicroPython, one would need to reflash the ESP32s with MicroPython. A simple script could be written that sets one ESP32 as a Transceiver, the one that sends data, and sets the other ESP32 as a Receiver, the one that only receives data. By creating a simple script that sends some kind of data from one ESP32 to the other, you can test the speed at which data can be transferred. An ESP32 should be used as the microcontroller of choice because of its capabilities in being able to communicate via Bluetooth or Wi-Fi. Through this process, one should be able to create the complete system we first set out to create.

# Appendix

## A. References


Baez, Emmanuel, et al. “Emnabz909/Pseudo_gps_for_romi_bots.” *GitHub*, Charlie Refvem, 2024, github.com/Emnabz909/Pseudo_GPS_for_Romi_Bots.

“Markers and Dictionaries.” *OpenCV*, docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Accessed 5 Dec. 2024.

Nayak, Sunita. “Augmented Reality Using ArUco Markers in OpenCV (C++ / Python): LEARNOPENCV #.” *LearnOpenCV*, 4 May 2021, learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/.

Politiek, Rients. “Install OpenCV on Raspberry Pi - Q-Engineering.” *Q*, Q-engineering, 29 Jan. 2024, qengineering.eu/install-opencv-on-raspberry-pi.html.

“Teachable Machine.” *Google*, Google, teachablemachine.withgoogle.com/train/image. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=0CoyNOVr91k. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=2vALTxu0yUc. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=6AY5p1uC5gM. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=Fchzk1lDt7Q. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=GpjX5MKIeao. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=LPC4ftpdc-4. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=V62M9d8QkYM. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=Vg9rrOFmwHo. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=Wl11eloYVm8. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=Z8cs1cRrc5A. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=_QpNMJEAkX0. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=aFNDh5k3SjU. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=bEKjCDDUPaU&list=PLWNDWPAClRVqNUIuJylljkQfFSeIpuxUi&index=5. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=dDkPB-dF1QA. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=iOTWZI4RHA8. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=kX6zWqMP9U4. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=kuJpdAf07WQ. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=qNdcXUEF7KU. Accessed 5 Dec. 2024.

YouTube. *YouTube*. www.youtube.com/watch?v=zxjhhzU_wkc. Accessed 5 Dec. 2024.

## B. Final Budget

C. **Table 6.** Budget Chart

| Pseudo GPS for Romi Final Budget | | |
|---|---|---|
| Part | Cost [$] | Purchased By |
| Micro HDMI to HDMI cable (3ft) | 8.93 | Sponsor |
| TwoZoh Micro HDMI to HDMI cable (15ft) | 13.99 | Sponsor |
| Raspberry Pi Camera Module 3 Standard - 12MP Autofocus | 25.00 | Sponsor |
| Raspberry Pi 4 model B-8GB RAM x2 | 150.00 | Sponsor |
| Official Raspberry Pi Power Supply 5.1V 3A with USB C | 7.95 | Sponsor |
| SD/MicroSD Memory Card - 16GB Class 10 - Adapter Included | 19.95 | Sponsor |
| Official Raspberry Pi 4 Case Fan and Heatsink | 5.00 | Sponsor |
| Twozoh 4K micro-HDMI to HDMI 60Hz 1080p | 13.99 | Sponsor |
| USB Charger Block OKRAY 2-pack 18W | 8.99 | Sponsor |
| Baiwwa USB C Cable, USB A to Type C Cable Charger | 8.98 | Sponsor |
| 1/4"-20 1" Machine Screws | 1.36 | Team |
| M2.5 10mm Machine Screws | 10.86 | Team |
| M2 10mm Machine Screws | 9.99 | Team |
| M2 Nuts | 0.00 | Team |
| M2.5 Threaded Inserts | 8.99 | Team |
| 1/8" Mushroom Head Toggle Bolt Anchors | 8.48 | Team |
| Overture Black & White PLA 2pack, 1kg spools | 28.99 | Team |
| **Parts Total** | 331.47 | |
| **Project Budget** | 500.00 | |

| Recommended Parts | | |
|---|---|---|
| Parts | Cost [$] | Purchased By |
| Raspberry Pi Screen, ROAMDOM 10.1" Touchscreen Monitor, 1024 x 600 | 99.99 | Team |
| Logitech MK270 Wireless Keyboard and Mouse Combo Beikell SD Card Reader, Dual Connector USB C Memory Card Reader Adapter | 27.99 | Team |
| Beikell SD Card Reader, Dual Connector USB C Memory Card Reader Adapter | 7.59 | Team |
| Parts Total | 7.59 | |
| Current Budget | 135.57 | |

## C. User Manual

# User Manual
## Pseudo GPS for Romi System

Team Members:
Emmanuel Baez, Gabriel Coria, Owen Guinane, Conor Schott

*ME Department, California Polytechnic State University San Luis Obispo*

December 8, 2024

**Abstract**

This user manual offers comprehensive guidance for Cal Poly Mechatronic students on setting up, operating, and maintaining the Pseudo GPS for the Romi system. Utilizing a ceiling-mounted camera, the system tracks and monitors the positions and orientations of Romi robots within the ME mechatronics lab environment.

# 1 Bill of Materials

| Romi Budget Plan | |
|---|---|
| | |
| Part | Cost $ |
| Micro HDMI to HDMI cable | 8.95 |
| Raspberry Pi Camera Module 3 Standard - 12MP Autofocus | 25 |
| Raspberry Pi 4 model B-8GB RAM x2 | 150 |
| Official Raspberry Pi Power Supply 5.1V 3A with USB C | 7.95 |
| SD/MicroSD Memory Card - 16GB Class 10 - Adapter Included | 19.95 |
| Official Raspberry Pi 4 Case Fan and Heatsink | 5 |
| Twozoh 4K micro HDMI to HDMI 60Hz 1080p | 13.99 |
| USB CHarger Block OKRAY 2-pack 18W | 8.99 |
| Baiwwa USB C Cable, USB A to type C Cable Charger | 8.98 |
| Parts Total | 248.81 |
| Initial Budget | 500 |
| Current Budget | 251.19 |

| Recommended Parts | |
|---|---|
| | |
| Part | Cost $ |
| Raspberry Pi Screen, ROAMDOM 10.1" Touchscreen Monitor, 1024 x 600 | 99.99 |
| Logitech MK270 Wireless Keyboard and Mouse COmbo | 27.99 |
| Beikell SD Card Reader, Dual Connector USB C Memory Card Reader Adapter | 7.59 |
| | |
| | |
| | |
| | |
| | |
| | |
| Parts Total | 135.57 |
| Current Budget | 251.19 |
| New Budget | 115.62 |

Figure 1: Bill of Materials Image

# 2 Software Setup

This section explains how to set up the software environment on the Raspberry Pi to run the Pseudo GPS for Romi system. It covers how to turn on the Raspberry Pi, install required libraries like OpenCV, and run the provided Python code to interact with the camera and perform ArUco marker detection.

## 2.1 Required Python Imports

To run the system, you'll need to install and import several Python libraries. Below are the imports required for the Thonny Python environment and the code functionality.

### 2.1.1 Required Imports

Include these imports at the beginning of your Python script:

```
import cv2 # For image processing and ArUco marker detection
import numpy as np # For handling arrays and matrix operations
from picamera2 import Picamera2, MappedArray  # For accessing the
    Raspberry Pi camera
import time # For managing time-related operations
import math # For mathematical operations like trigonometry
import os # For handling system-related operations
import sys # For system-specific parameters
import threading # For managing concurrent operations
import argparse # For parsing command-line arguments
from cv2 import aruco # For working with ArUco marker detection
```

These imports are necessary for the core functionalities of the Pseudo GPS system:

- `cv2`: OpenCV, used for image processing and computer vision tasks like ArUco marker detection.
- `numpy`: For handling matrix and array-based computations.
- `picamera2`: To interface with the Raspberry Pi camera.
- `time`: For managing delays or intervals, like timing for marker detection.
- `math`: For handling mathematical operations such as angles or trigonometry.
- `os`, `sys`: For system operations, managing file paths, and handling the environment.
- `threading`: If you're working with concurrent tasks.
- `argparse`: For parsing command-line arguments if necessary.
- `aruco`: For specific functions related to ArUco markers.

# 3 Installing Dependencies

To run the provided Python code, you will need to install OpenCV, Picamera2, NumPy, and other libraries. Below are the installation steps.

## 3.1 Install OpenCV and Required Libraries

Use the terminal on your Raspberry Pi to install the necessary libraries.

### 3.1.1 Update the Raspberry Pi

To Update the Pi, you may need this info:

username for Pi: emanbz909
Password: BigRed405
this is for main Pi in black housing clone of pi is the white housing
username for clone Pi: emanbz909 (@CodyClone1)

Open a terminal on the Raspberry Pi and run the following command to update the package lists:

```
sudo apt-get update
sudo apt-get upgrade
```

### 3.1.2 Install Dependencies

Install the required dependencies for OpenCV, camera, and additional libraries:

```
sudo apt-get install python3-opencv
sudo apt-get install python3-picamera2
sudo apt-get install python3-numpy
sudo apt-get install python3-matplotlib
```

### 3.1.3 Install OpenCV via pip (if not already installed)

If OpenCV is not already installed via the default repositories, you can install it using pip. Note: This may take around one hour to install.

```
pip3 install opencv-python
pip3 install opencv-contrib-python
```

# 4 Hardware Setup

This section explains how to physically set up the hardware components of the Pseudo GPS for Romi system. This includes assembling the ceiling-mounted camera, installing the Raspberry Pi, and other components, and how the system works together to track the Romi robots.

## 4.1 Hardware Components

- **Raspberry Pi 4 Model B (1X)**: The main processing unit responsible for controlling the system and processing camera data.
- **Raspberry Pi Camera Module 3 (1X)**: A high-resolution camera mounted on the ceiling to capture the environment and track the Romi robots.
- **ESP32 Microcontroller (2X)**: Used for wireless communication between the Raspberry Pi and the Romi robots via WiFi.
- **Mounting Hardware**: Includes screws, threaded inserts, and toggle bolts for securely attaching the system to the ceiling.

## 4.2 Setting Up the Camera

### 4.2.1 Camera Installation

- Mount the Raspberry Pi Camera Module 3 securely to the ceiling using the camera housing.
- Ensure that the camera is properly aligned to provide an optimal field of view of the area where the Romi robots will be tracked.
- The camera is connected to the Raspberry Pi through the CSI (Camera Serial Interface) port.

### 4.2.2 Camera Calibration

The camera calibration process uses ArUco markers placed in the environment. The system estimates their pose (position and orientation) relative to the camera using the calibration matrix and distortion coefficients. You will need to adjust the **calibration factor** (located in our code) if the detected distances between markers are incorrect. This calibration must be done each time you change the camera's height distance from the arUco markers.

**Resource:**

- Camera Calibration Example: OpenCV Interactive Calibration

## 4.3 Hardware Setup Visual Images

Here are placeholder images to represent the hardware components. These images will be replaced with actual pictures once they are available.

### 4.3.1 Raspberry Pi 4 Model B with Mounting Hardware

Figure 2: Raspberry Pi 4 Model B in mounting hardware.

### 4.3.2 Raspberry Pi Camera Module 3

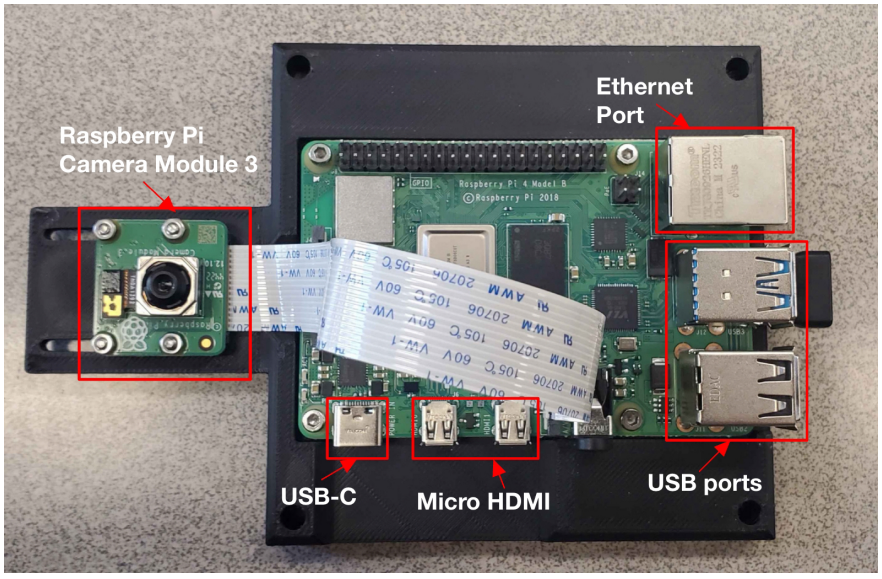



Figure 3: Labeled Raspberry Pi Camera Module 3 Top View

### 4.3.3 Raspberry Pi Camera Module 3

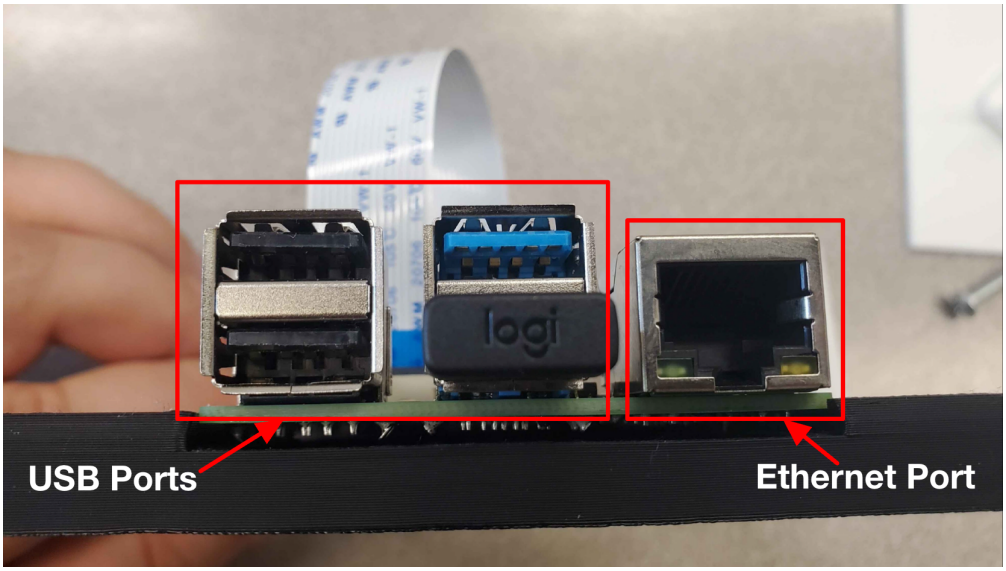



Figure 4: Labeled Raspberry Pi Camera Module 3 Back View

### 4.3.4 Raspberry Pi Camera Module 3

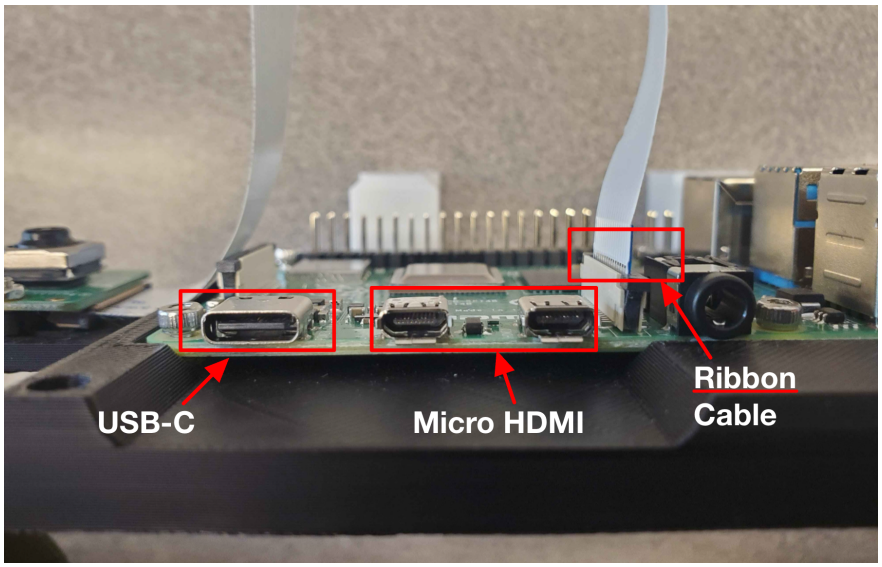



Figure 5: Labeled Raspberry Pi Camera Module 3 Side View

### 4.3.5 Raspberry Pi Camera Module 3

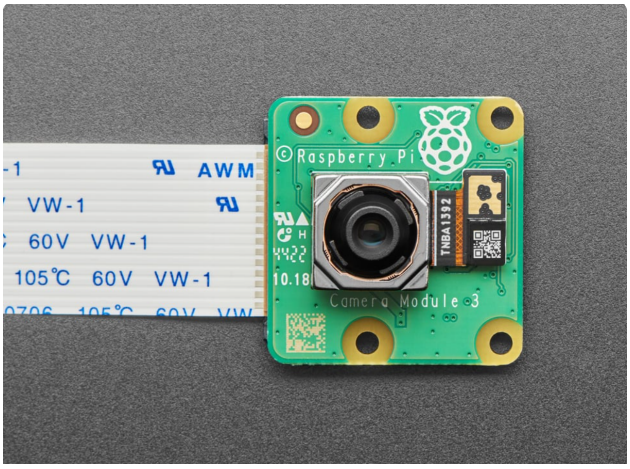



Figure 6: Raspberry Pi Camera Module 3

### 4.3.6 ESP32 Microcontroller

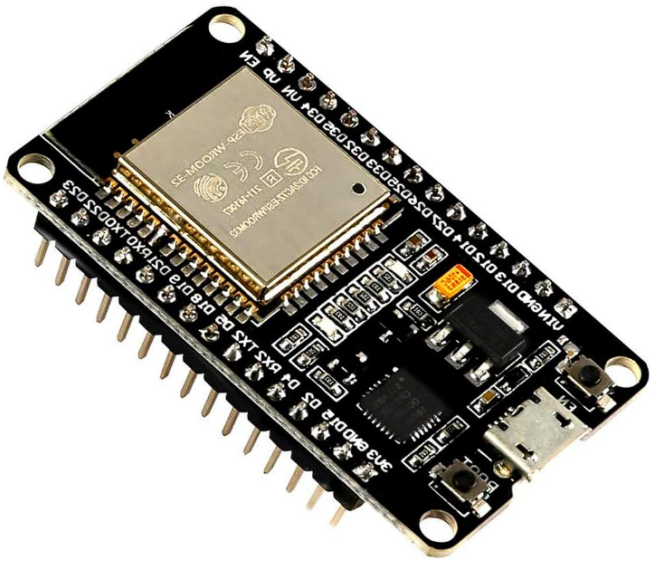


Figure 7: ESP32 Microcontroller

# 5 Code Walkthrough

Below is an explanation of the provided Python code, which runs on the Raspberry Pi to track and monitor the Romi robots using ArUco markers. The code performs the key functions of capturing images from the camera, detecting ArUco markers, and estimating their pose (position and orientation). Note: the orientation and position codes will not run until an ArUco marker is placed. The marker generator code only generates markers.

## 5.1 Initial Setup and Camera Class

```
import cv2
import numpy as np
from picamera2 import Picamera2, MappedArray
import time
import math
import os
import sys
import threading
import argparse
from cv2 import aruco

# Initialize the camera
class Camera:
    def __init__(self):
        self.picam2 = Picamera2()
        self.picam2.start()

    def capture_frame(self):
        frame = self.picam2.capture_array()
        return frame
```

### 5.1.1 Explanation:

1. **Imports**: The code begins by importing the necessary libraries. These include OpenCV (`cv2`), NumPy (`np`), Picamera2 for interfacing with the Raspberry Pi camera, and others for threading and argument parsing.

2. **Camera Class**: - A `Camera` class is defined that initializes the Pi camera (`Picamera2`) and starts capturing frames.

- The method `capture_frame()` captures a single frame from the Raspberry Pi camera and returns it as a NumPy array for further image processing.

## 5.2 Marker Detection and Pose Estimation

```
# Marker detection and pose estimation
def detect_markers(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    detector = aruco.ArucoDetector()
    markers, ids, _ = detector.detectMarkers(gray)
    return markers, ids
```

### 5.2.1 Explanation:

1. **detect_markers()**: - Converts the captured frame to grayscale using `cv2.cvtColor`. - Initializes an ArUco marker detector using `aruco.ArucoDetector()`. - Detects ArUco markers in the grayscale image using the `detectMarkers()` method, which returns the marker positions (`markers`) and their IDs (`ids`).

## 5.3 Main Program Loop

```
# Main program loop
if __name__ == "__main__":
    camera = Camera()
    while True:
        frame = camera.capture_frame()
        markers, ids = detect_markers(frame)

        if markers:
            for marker in markers:
                cv2.drawContours(frame, marker, -1, (0, 255, 0), 2)

        cv2.imshow("Camera Feed", frame)
        if cv2.waitKey(1) == ord('q'):
            break

    cv2.destroyAllWindows()
```

### 5.3.1 Explanation:

1. **Main Loop**:
   - The camera captures a frame in an infinite loop (`while True:`).
   - For each frame, the code checks for ArUco markers and draws contours around the detected markers.
   - The frame is displayed using OpenCV's `imshow()`, and the loop continues until the user presses 'q'.
2. **Exit**:

- The loop breaks when 'q' is pressed, and the window is destroyed using `cv2.destroyAllWindows()`.
- `cv2.destroyAllWindows()` is called to clean up and close any open OpenCV windows.

# 6 Preliminary Tests

Before using the system for real-time tracking, it's important to perform some preliminary tests to ensure everything is working correctly.

## 6.1 Test 1: Camera Functionality

- Open Thonny or Visual Studio Code and run a simple Python script to capture a single image from the camera.
- Ensure the camera feed is clear and the image appears on the screen.
- If the image is blurry or the camera feed does not appear, check the camera settings and calibration.

## 6.2 Test 2: ArUco Marker Detection

- Run the marker generation script to create a set of ArUco markers.
- Place the generated markers in the camera's field of view.
- Run the marker pose estimation script and verify that the markers are detected and their position is accurately displayed.
- If the markers are not detected, check the camera alignment and lighting conditions in the environment.

## 6.3 Test 3: Communication with ESP32

- Ensure that both the Raspberry Pi and the ESP32 are connected to the same Wi-Fi network.
- Run the code that sends position data from the Raspberry Pi to the ESP32.
- Verify that the ESP32 is receiving and processing the data correctly by using serial output or any other debug method.

## 6.4 AruCo Markers: A Brief Introduction

ArUco markers are square-shaped fiducial markers used in computer vision, particularly for tasks like augmented reality and camera pose estimation. Each marker contains a unique ID encoded in a binary grid, typically consisting of black and white squares. The markers are easily detectable due to their high contrast and distinctive pattern, even when rotated or partially occluded. In OpenCV, ArUco markers are detected using the `cv2.aruco` module, which can identify the marker's position and orientation in an image. The detection process involves converting the image to grayscale, applying an algorithm to locate the marker's corners, and then decoding the ID. OpenCV can also estimate the 3D pose of the marker relative to the camera if the camera calibration data is available, making ArUco markers ideal for tracking and overlaying virtual objects in real-world environments.

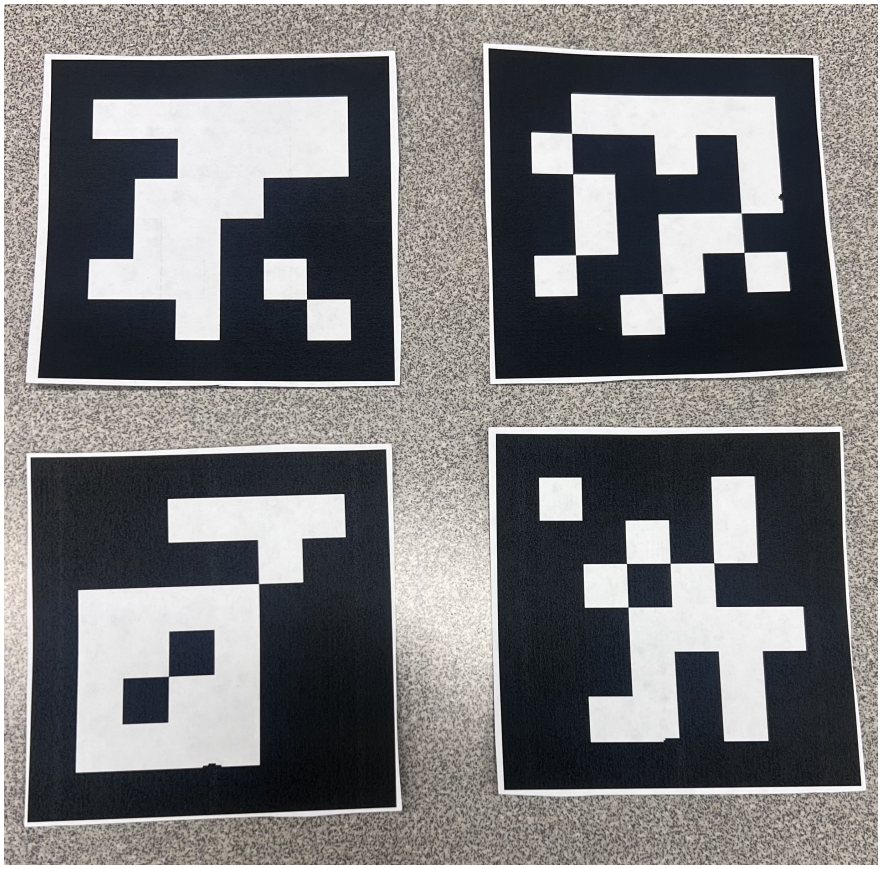


Figure 8: Sample ArUco Markers

# 7 Conclusion

The Pseudo GPS for Romi system is a robust solution for tracking the positions and orientations of Romi robots within a lab environment. By utilizing a ceiling-mounted camera and ArUco marker detection, this system offers a proof of concept for precise real-time localization of robots.

## D. Risk Assessment

## designsafe Report

| | | | |
|---|---|---|---|
| Application: | Romi GPS | Analyst Name(s): | Emmanuel Baez, Gabriel Coria, Owen Guinane, Conor Schott |
| Description: | This analysis concerns the entire Romi GPS system and associated assemblies, subassemblies, and components | Company: | Cal Poly |
| Product Identifier: | | Facility Location: | San Luis Obispo, California, USA |
| Assessment Type: | Detailed | | |
| Limits: | Analysis | | |
| Sources: | personnel experiences, ANSI B11 standards, assembly drawings W-Z | | |
| Risk Scoring System: | ANSI B11.0 Two Factor | | |

Guide sentence: When doing [task], the [user] could be injured by the [hazard] due to the [failure mode].

| Item Id | User / Task | Hazard / Failure Mode | Initial Assessment Severity Probability | Risk Level | Risk Reduction Methods /Control System | Final Assessment Severity Probability | Risk Level | Status / Responsible /Comments /Reference |
|---|---|---|---|---|---|---|---|---|
| 1-1-1 | operator<br>normal operation | slips / trips / falls : falling material / object<br>Loose fasteners, Improper installation, Tile wear | Serious<br>Unlikely | Medium | prevent energy release<br>/Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Owen Guinane |
| 1-2-1 | operator<br>load / unload materials | mechanical : impact<br>Dropped parts | Minor<br>Likely | Low | instruction manuals<br>/Not Applicable | Minor<br>Likely | Low | TBD<br>Gabriel Coria |
| 1-2-2 | operator<br>load / unload materials | electrical / electronic : energized equipment / live parts<br>Improper wiring | Minor<br>Unlikely | Negligible | instruction manuals<br>/Not Applicable | Minor<br>Unlikely | Negligible | TBD<br>Emmanuel Baez |
| 1-2-3 | operator<br>load / unload materials | slips / trips / falls : fall hazard from elevated work<br>User must be elevated to install/ remove | Serious<br>Unlikely | Medium | instruction manuals<br>/Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Conor Schott |
| 1-2-4 | operator<br>load / unload materials | slips / trips / falls : falling material / object<br>Loose fasteners, Improper installation, Tile wear | Serious<br>Unlikely | Medium | prevent energy release<br>/Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Owen Guinane |
| 1-3-1 | operator<br>clean system | electrical / electronic : energized equipment / live parts<br>Improper wiring | Minor<br>Unlikely | Negligible | instruction manuals<br>/Not Applicable | Minor<br>Unlikely | Negligible | TBD<br>Emmanuel Baez |
| 1-3-2 | operator<br>clean system | slips / trips / falls : fall hazard from elevated work<br>User must be elevated to install/ remove | Serious<br>Unlikely | Medium | instruction manuals<br>/Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Conor Schott |

| Item Id | User / Task | Hazard / Failure Mode | Initial Assessment Severity Probability | Risk Level | Risk Reduction Methods /Control System | Final Assessment Severity Probability | Risk Level | Status / Responsible /Comments /Reference |
|---|---|---|---|---|---|---|---|---|
| 2-1-1 | maintenance technician periodic maintenance | mechanical : impact Dropped parts | Minor Likely | Low | instruction manuals /Not Applicable | Minor Likely | Low | TBD Gabriel Coria |
| 2-1-2 | maintenance technician periodic maintenance | electrical / electronic : energized equipment / live parts Improper wiring | Minor Unlikely | Negligible | instruction manuals /Not Applicable | Minor Unlikely | Negligible | TBD Emmanuel Baez |
| 2-1-3 | maintenance technician periodic maintenance | slips / trips / falls : falling material / object Loose fasteners, Improper installation, Tile wear | Serious Unlikely | Medium | prevent energy release /Not Applicable | Serious Unlikely | Medium | TBD Owen Guinane |
| 2-2-1 | maintenance technician trouble-shooting / problem solving | mechanical : impact Dropped parts | Minor Likely | Low | instruction manuals /Not Applicable | Minor Likely | Low | TBD Gabriel Coria |
| 2-2-2 | maintenance technician trouble-shooting / problem solving | electrical / electronic : energized equipment / live parts Improper wiring | Minor Unlikely | Negligible | instruction manuals /Not Applicable | Minor Unlikely | Negligible | TBD Emmanuel Baez |
| 2-2-3 | maintenance technician trouble-shooting / problem solving | slips / trips / falls : fall hazard from elevated work User must be elevated to install/ remove | Serious Unlikely | Medium | instruction manuals /Not Applicable | Serious Unlikely | Medium | TBD Conor Schott |
| 2-2-4 | maintenance technician trouble-shooting / problem solving | slips / trips / falls : falling material / object Loose fasteners, Improper installation, Tile wear | Serious Unlikely | Medium | prevent energy release /Not Applicable | Serious Unlikely | Medium | TBD Owen Guinane |
| 2-3-1 | maintenance technician installation | mechanical : impact Dropped parts | Minor Likely | Low | instruction manuals /Not Applicable | Minor Likely | Low | TBD Gabriel Coria |
| 2-3-2 | maintenance technician installation | electrical / electronic : energized equipment / live parts Improper wiring | Minor Unlikely | Negligible | instruction manuals /Not Applicable | Minor Unlikely | Negligible | TBD Emmanuel Baez |

| Item Id | User / Task | Hazard / Failure Mode | Initial Assessment Severity Probability | Risk Level | Risk Reduction Methods /Control System | Final Assessment Severity Probability | Risk Level | Status / Responsible /Comments /Reference |
|---|---|---|---|---|---|---|---|---|
| 2-3-3 | maintenance technician installation | slips / trips / falls : fall hazard from elevated work<br>User must be elevated to install/ remove | Serious<br>Unlikely | Medium | instruction manuals /Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Conor Schott |
| 2-3-4 | maintenance technician installation | slips / trips / falls : falling material / object<br>Loose fasteners, Improper installation, Tile wear | Serious<br>Unlikely | Medium | prevent energy release /Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Owen Guinane |
| 3-1-1 | passer-by / non-user walk near machinery | mechanical : impact<br>Dropped parts | Minor<br>Likely | Low | instruction manuals /Not Applicable | Minor<br>Likely | Low | TBD<br>Gabriel Coria |
| 3-1-2 | passer-by / non-user walk near machinery | slips / trips / falls : falling material / object<br>Loose fasteners, Improper installation, Tile wear | Serious<br>Unlikely | Medium | prevent energy release /Not Applicable | Serious<br>Unlikely | Medium | TBD<br>Owen Guinane |

# Design Failure Mode and Effects Analysis

Product: _____ Prepared by: _____

Team: _____ Date: _____ (orig)

| | | | | | | | | | | | | Action Results | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| System / Function | Potential Failure Mode | Potential Effects of the Failure Mode | Severity | Potential Causes of the Failure Mode | Current Preventative Activities | Occurence | Current Detection Activities | Detection | RPN | Recommended Action(s) | Responsibility & Target Completion Date | Actions Taken | Severity | Occurence | Detection | RPN |
| Bracket/supports weight of system | Bracket Breaks | a) System collapses<br>b) Sharp edges exposed<br>c) User is injured<br>d) Components damaged | 9 | 1) Bracket is too thin<br>2) System is too heavy<br>3) Repeated loading/unloading | 1) Measure weight of system components<br>2) Stress analysis<br>3) Fatigue strength | 1 | Structural test | 4 | 36 | | | | | | | |
| | Bracket Flexes | a) Bracket feels unstable | 6 | 1) Bracket is too thin<br>2) Bracket geometry not rigid enough | 1) Impact factor<br>2) Stress analysis<br>3) Fatigue strength<br>4) Deflection analysis | 2 | Structural Test | 2 | 24 | | | | | | | |
| | Fastener Breaks | a) System collapses<br>b) Sharp edges exposed<br>c) User is injured<br>d) Components damaged | 9 | 1) Not enough fasteners<br>2) Fasteners used too short | 1) Stress analysis<br>2) Fatigue strength | 1 | Structural Test | 5 | 45 | | | | | | | |
| Bracket/ fits securely to roof | Bracket Moves | a) Bracket feels unstable<br>b) Interferes with system calibration | 6 | 1) Guideways designed incorrectly<br>2) Guideways printed incorrecty | 1) Deflection analysis<br>2) Roof dimension analysis<br>3) Redesign guides | 4 | Fitment check | 1 | 24 | | | | | | | |
| Housing/ Fits all our components | Housing Breaks | a) System collapses<br>b) Sharp edges exposed<br>c) User is injured<br>d) Components damaged | 6 | 1) Geometry was off<br>2) Used wrong material | 1) stress analysis<br>2) fatigue strength | 2 | Structural test | 4 | 48 | | | | | | | |
| | Housing Flexes | a) housing feels unstable | 6 | 1) Geometry was off<br>2) Used wrong material | 1) stress analysis<br>2) fatigue strength | 2 | Structural test | 2 | 24 | | | | | | | |
| Microcontroller/ interpret data | Software error | a) Tracking inaccuraries,<br>b) Full system crash | 5 | 1) Program bugs | 1) Utilize program debugger<br>2) Update software<br>3) Delete corrupted files | 6 | 1) Bug testing/ System testing | 8 | ## | | | | | | | |
| Electronics/ System Power | Hardware error | a) System is overloaded with voltage/current | 8 | 1) Inadequete surge protectors<br>2) Improper wiring<br>3) Defective hardware | 1) Surge Protector<br>2) Proper wiring<br>3) Test initial hardware | 1 | Voltameter test | 6 | 48 | | | | | | | |
| Microcontroller/ communicate information to Romi | Connection loss | a) Robots lose accurate tracking information | 5 | 1) Software error<br>2) Modem failure (Wi-fi)<br>3) EM Interference | 1) Limit interference | 4 | System test | 2 | 40 | | | | | | | |
| | Slow connection | a) Large latency in streaming tracking data to robots | 4 | 1) Software error<br>2) Modem failure (Wi-fi)<br>3) EM Interference | 1) Limit interference | 4 | System test | 2 | 32 | | | | | | | |



Revision Date:

# Design Failure Mode and Effects Analysis

Product: _____ Prepared by: _____

Team: _____ Date: _____ (orig)

| | | | | | | | | | | | | Action Results | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| System / Function | Potential Failure Mode | Potential Effects of the Failure Mode | Severity | Potential Causes of the Failure Mode | Current Preventative Activities | Occurence | Current Detection Activities | Detection | RPN | Recommended Action(s) | Responsibility & Target Completion Date | Actions Taken | Severity | Occurence | Detection | RPN |
| Camera/tracking Romis and obstacles | Resolution error | a) Camera cannot track Romis accurately<br>b) Camera cannot communicate data to microcontroller | 7 | 1) System is heavily outdated<br>2) System's specs don't satisfy our requirements | 1) Hardware Calibration | 5 | Hardware test | 3 | ## | | | | | | | |



Revision Date:

## E. Test Procedures & Results

**Test Name**: Tracking Resolution Test

**Purpose: Test** to determine how accurately the camera locates Romi bots

**Scope:** This test will determine the tracking accuracy of a single Romi robot.

**Equipment:** Final assembly (10), Lab table, Grid mat, Meter stick

**Hazards:** No Hazards Present

**PPE Requirements**: None

**Facility:** Mechatronics lab

**Procedure:**

1. Wire all components together and provide power to the Pi and Camera.
2. Measure and record the distance between the top left and bottom right corners of the mat on the lab table using measuring tape.
3. Load and run the main code for the project.
4. Check to see if the Camera recognizes the origin (ArUco Marker 1).
5. Wait until the program is steadily tracking the origin.
6. Place six ArUco markers at arbitrary points on the grid mat.
7. Record the coordinates of the marker displayed on the program.
8. Measure the distances from center to center of ArUco markers to check accuracy.
9. Remove the markers from the spot on the grid map. Determine the center point of the ArUco marker from the marks on the grid mat.
10. Measure and record the distance of the ArUco marker center.
11. Repeat steps 7-10 for a total of five runs for a baseline height of z=0 mm.
12. Next, place a Romi with an ArUco marker on a random point on the grid map.
13. Record the coordinates of the marker displayed on the program.
14. Measure the distances from center to center of the Romi to check accuracy.
15. Remove the Romi from the spot on the grid map. Determine the center point of the ArUco marker from the marks on the grid mat.
16. Measure and record the distance of the ArUco marker center.
17. Repeat steps 12-16 for a total of five runs for a baseline height of the Romi, z=105mm.
18. Next, an ArUco marker at a various height (on top of a roll of tape, pencil box, just the table Romi, etc.) on a random point on the grid map
19. Record the coordinates of the marker displayed on the program.
20. Measure the distances from center to center of the ArUco marker to check accuracy.
21. Remove the markers from the spot on the grid map. Determine the center point of the ArUco marker from the marks on the grid mat.
22. Measure and record the distance of the ArUco marker center.
23. Repeat steps 18-22 for a total of five runs for a baseline height of the Romi, z=105mm.

**Results:** To pass the test, the system must successfully detect the origin (ArUco marker 1) and have a calculated tracking accuracy of ± 5mm or lower for 80% of all trials conducted. Otherwise, the system fails.

Uncertainty analysis should consider the resolution uncertainty of the meter stick as well as the resolution uncertainty.

$$Ures = sqrt({\sigma_t}^2 + {\sigma_m}^2)$$

| **Tracking Resolution Test** | | | | |
|---|---|---|---|---|
| *Constant Height Resolution Test at z = 0 mm* | | | | |
| | | Computer [mm] | Actual [mm] | Abs. Difference [mm] |
| Run 1 | X | 7 | 0 | 7 |
| | Y | 362 | 361 | 1 |
| Run 2 | X | 89 | 90 | 1 |
| | Y | 2 | 0 | 2 |
| Run 3 | X | 139 | 137 | 2 |
| | Y | 46 | 42 | 4 |
| Run 4 | X | 35 | 30 | 5 |
| | Y | 236 | 242 | 6 |
| Run 5 | X | 455 | 443 | 12 |
| | Y | 248 | 255 | 7 |
| Average Tracking Resolution [mm] | | | | 4.70 |
| Standard Deviation [mm] | | | | 3.47 |
| *Constant Height Resolution Test at z = 105 mm (Romi)* | | | | |
| Run 1 | X | 310 | 296 | 14 |
| | Y | 95 | 85 | 10 |
| Run 2 | X | 192 | 178 | 14 |
| | Y | 339 | 330 | 9 |
| Run 3 | X | 232 | 225 | 7 |
| | Y | 144 | 150 | 6 |
| Run 4 | X | 208 | 195 | 13 |
| | Y | 231 | 220 | 11 |
| Run 5 | X | 141 | 135 | 6 |
| | Y | 232 | 215 | 17 |
| Average Tracking Resolution [mm] | | | | 10.70 |
| Standard Deviation | | | | 3.77 |

| *Variable-Height Resolution test* | | | | |
|---|---|---|---|---|
| Run 1 (Z = 15mm) | X | 86 | 81 | 5.00 |
| | Y | 369 | 370 | 1.00 |
| Run 2 (Z = 113mm) | X | 605 | 590 | 15.00 |
| | Y | 202 | 190 | 12.00 |
| Run 3 (Z = 50mm) | X | 536 | 534 | 2.00 |
| | Y | 378 | 380 | 2.00 |
| Run 4 (Z = 113mm) | X | 24 | 35 | 11.00 |
| | Y | 53 | 60 | 7.00 |
| Run 5 (Z = 50mm) | X | 117 | 122 | 5.00 |
| | Y | 460 | 463 | 3.00 |
| Average Tracking Resolution [mm] | | | | 6.30 |
| Standard Deviation | | | | 4.83 |
| *Overall Results* | | | | |
| Overall Avg. Tracking Resolution [mm] | | | | 7.23 |
| Resolution Uncertainty [mm] | | | | 7.04 |

Calibration variables

Calibration factor = 2.754

$A_x = 0.05$

$A_y = 0.65$

Marker Size = 3.5 (in)

Test Date(s): 11/9, 11/15

**Test Result**s: Fail

Performed By: Emmanuel Baez, Gabriel Coria, Owen Guinane, Conor Schott

**Test Name**: Simultaneous Detection/Swarming Test

**Purpose:** Test to determine if the system can detect multiple Romi bots simultaneously

**Scope:** This test will determine the amount of time that the program can correctly identify multiple Romi bots simultaneously.

**Equipment:** Final Assembly (0), Romi Robots

**Hazards:** No Hazards Present

**PPE Requirements**: None

**Facility:** Mechatronics lab

**Procedure:**

1. Ensure that the final assembly (0) is installed properly and powered.
2. Load and run the main code for the project.
3. Check to see if the Camera recognizes the origin (ArUco Marker 1).
4. Wait until the program is steadily locked on origin.
5. Place multiple Romis with ArUco markers at random spots on the table.
6. Record the coordinates of the Romi robot displayed on the program.
7. Measure the distances from center to center of ArUco markers to check accuracy.
8. Move the Romi robots to random positions on the mat.
9. Repeat steps 8-9 for a total of 15 trials.

**Results:** The system passes if it can detect and uniquely identify all six Romi robots for at least 80% of trials conducted. The system fails otherwise.

Uncertainty analysis is not required for this test.

Test Date(s): 11/15

Test Results: Pass

Performed By: Emmanuel Baez, Gabriel Coria, Owen Guinane, Conor Schott

**Test Name**: Ceiling Test

**Purpose:** Test to determine if the system can safely be attached to the ceiling

**Scope:** The best way to get a view of the Romi robots' operating area is to mount the system vertically above the lab table. To accomplish this, the system must be able to safely be mounted on a ceiling tile overseeing the operating area

**Equipment:** Main housing (210), Ceiling tile (110)

**Hazards:** Fall Hazards present

**PPE Requirements**: Eyeglasses

**Facility:** Mechatronics lab

**Procedure:**

1. Secure the main housing (210) to the ceiling tile (110).
2. Replace one of the ceiling tiles from the lab with the modified ceiling tile.
3. Monitor the system to observe if the system will fall or sag and monitor for a 3-hour period (simulate lab).
4. Repeat steps 1-3 for a total of three trials.

**Results:** Results are based on Pass/Fail criteria. The system passes if the system does not fall and sags and fails if it does.

Uncertainty analysis is not required for this test.

Test Date(s): 11/15

Test Results: Pass

Performed By: Owen Guinane

**Test Name**: Fitment & Weight Test

**Purpose: Test** to determine that the weight we are adding to the Romi robots will not encumber them. We are adding two attachments to the Romi bots to enable orientation data collection and facilitate data transfer between the bots and the system. The first attachment is a large shape mounted on top of the Romi bots, designed for easy identification by the camera. The second attachment is an ESP32 module, which will serve as a transceiver for data communication.

**Scope: To** assess the impact of attachments, we will test the motors' ability to handle the extra weight by having the Romi bots run and monitoring their performance. We will also test the stability of the additional parts on the Romi bots. During the tests, we will monitor any signs of looseness while the bots are in motion.

**Equipment:** Romi bot, Hat attachment, ESP32

**Hazards:** No Hazards Present

**PPE Requirements**: None

**Procedure:**

1. Attach all external components to the Romi.
2. Run a simple program on the Romi.
3. As the program is running, record if the Romi shows any sign of being encumbered or if accessories become unstable during operation.

**Results:** Results are based on a Pass/Fail criteria. The system passes if Romi can run unencumbered with all boards and attachments installed. The system fails if the Romi shows any signs of being encumbered or if any of the components detach during operation.

Uncertainty analysis should not be required for this test.

**Test Result**s: We were unable to run the test due to time limitations, as it was considered a lower priority. Had we had more time, we would have followed the procedures to determine whether the test passed or failed.

**Test Name**: Transfer Rate Test

**Purpose:** This test is designed to measure the speed at which positional data is transferred from the Raspberry Pi system to the Romi bots. Ensuring a reliable and high-frequency data transfer rate is critical for enabling real-time navigation and obstacle avoidance.

**Scope:** The scope of this test is to evaluate the efficiency and responsiveness of the data transmission process within the system. Accurate and timely position tracking of the Romi bots is essential to allow Mechatronics students to write and implement control algorithms that depend on real-time location data. The test specifically assesses whether the system can meet the minimum data transfer rate of 10Hz, which ensures that the Romi bots receive location updates quickly enough for real-time control in a dynamic environment. This test directly supports the overall project by confirming that the system's transfer rate is sufficient for practical, classroom-based robot tracking and interaction.

**Equipment:** Raspberry Pi, Pi Camera, Romi with ESP32

**Hazards:** No Hazards Present

**PPE Requirements**: None

**Facility:** Mechatronics lab

**Procedure:**

1. Wire all components together and provide power to the Raspberry Pi and camera.
2. Load and run the main code for the project.
3. Confirm that the camera can capture its intended view.
4. Place the ArUco marker that will be the origin in the camera's view.
5. Place a Romi with an ArUco marker in an arbitrary position on the table.
6. Upload and run a simple code that displays Romi’s position in the terminal of Thonny.
7. Measure and record the rate at which the Romi bot receives data.

**Results:** Results are based on a Pass/Fail criteria. The system passes if the data can transfer at a rate of 5Hz and fails if it cannot.

Uncertainty analysis should consider uncertainty in transfer rate resolution, accounting for potential timing delays and environmental factors that could affect measurement accuracy.

Test Results: We were unable to run the test due to the time limitations of setting up the ESP32, as it was considered a lower priority. Had we had more time, we would have followed the procedures to determine whether the test passed or failed.

**Test Name**: Orientation Accuracy test

**Purpose:** Test to determine if the program can accurately identify the heading of a Romi robot

**Scope:** The purpose of our project is to be able to track the position of the Romi bots so that the data can be immediately used by the Mechatronic students in their coding.

**Equipment:** Final Assembly [0], Romi with ESP32 and Hat Attachment, orientation calibration mat

**Hazards:** No Hazards Present

**PPE Requirements**: None

**Facility:** Mechatronics lab

**Procedure:**

1. Power on the system.
2. Ensure that the camera detects the origin (ArUco marker 1 or whatever marker you decide to choose).
3. Place the Romi robot with an ArUco marker on top at an arbitrary point on the table. Ensure the front is facing north on the mat.
4. Record the heading that the Romi bot is facing by looking at the terminal and camera feed.
5. Record what heading the program indicates the Romi robot is facing (Measured).
6. Turn the Romi robot 5 degrees counterclockwise.
7. Repeat steps 4-6 until the Romi robot has done one full 360-degree turn.

**Results:** The system passes if the accuracy of the measured heading is within ±1 degree of its true heading.

Uncertainty analysis should consider the resolution of the calibration mat as well as the resolution uncertainty of the program reporting the data. As accuracy is calculated by

$$Uhead = sqrt({\sigma_t}^2 + {\sigma_m}^2)$$

**Test Result**s: We could not complete this test due to time constraints, as it was a lower priority in our project timeline. Had we been able to conduct the test, we would have followed the outlined procedure.