The “MazeRunner” Robotic Maze Solving System:

Technical Documentation and User Manual

41069 Robotics Studio 2

Project Contributors:

Hugh Radvan – 13549340

Cristian Corso – 13549237

Ryan Thomas – 13938802

Nicholas Uremovic – 14174268

Table of Contents

1. Project Overview 3

2. Dependencies 4

2.1. Required Hardware 4

2.2. Required Software 4

3. Installation Guide 6

3.1. Hardware Installation 6

3.2. Software Installation 9

4. Operation Guide 10

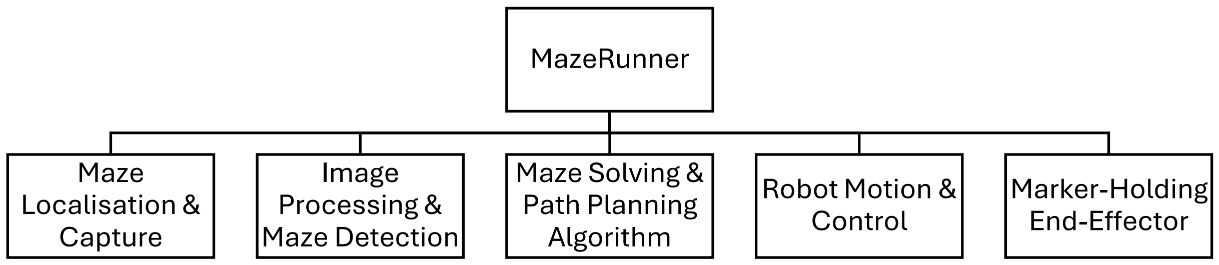
5. Subsystem Breakdown 13

6. Troubleshooting & FAQs 14

1. Project Overview

The MazeRunner project is a robotics system designed to autonomously solve and navigate a user defined maze using maze localisation, image processing, path planning, and motion control. The system captures an image of the maze, locates it in a global frame, processes it to extract walls vs. paths, computes an optimal solution, and physically traces the solution using a marker-holding end-effector.

System Architecture


MazeRunner
Maze Localisation & Capture
Image Processing & Maze Detection
Maze Solving & Path Planning Algorithm
Robot Motion & Control
Marker-Holding End-Effector

Key Features & Subsystems

- **Maze Localisation**: Captures a maze image using velocity-based control, guided by ArUco markers and live camera feed.
- **Image/Maze Processing**: Converts the captured maze image into a structured array representation for path planning.
- **Path Planning Algorithm**: Solves the maze to find an optimal path from start to finish using BFS and generates a list of cartesian waypoints for robot navigation.
- **Robot Control**: Executes trajectory planning based on waypoints and enables waypoint-based end-effector control via camera feed for image capture.
- **Marker-Holding End-Effector**: Secures both the marker and the camera to the robot end-effector.

2. Dependencies

2.1. Required Hardware

- <u>Connectivity</u>
    - PC Running Ubuntu 20.04.6 LTS (Focal Fossa)
    - USB-C to USB-3.0 cable
    - Ethernet cable
- <u>Fasteners</u>
    - 2x M3 bolts
    - 4x M6 bolts
- <u>Operation Systems</u>
    - UR3 Robot with network connectivity and installed ur_ros_driver
    - Intel RealSense d435i RGB-D camera
- <u>Maze Runner Accessories</u>
    - Custom marker holding UR3 end effector
    - Custom RealSense camera mount
    - Erasable white board marker
    - Laminated maze grid template

2.2. Required Software

- ROS Noetic: https://wiki.ros.org/noetic/Installation/Ubuntu

```bash
        $ sudo apt update
        $ sudo apt install ros-noetic-desktop-full
        $ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
        $ source ~/.bashrc
        $ sudo apt install python3-rosdep
        $ sudo rosdep init
        $ rosdep update
```

- Create Catkin workspace: We have called ours ws_moveit, however you may change the name. (If you chose to change the name, ensure that all instances of ws_moveit in the instructions are tailored to the name of your new workspace)

```bash
        $ sudo apt install python3-catkin-tools
        $ mkdir -p ~/ws_moveit/src
        $ cd ~/ws_moveit/
        $ catkin init
        $ catkin config –extend /opt/ros/noetic
        $ echo source ~/ws_moveit/devel/setup.bash
        $ source ~/.bashrc
```

- Install MoveIt:
  https://moveit.github.io/moveit_tutorials/doc/getting_started/getting_started.html

```bash
        $ sudo apt install ros-noetic-moveit
```

- Universal Robots ROS Driver: https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

```bash
$ source /opt/ros/noetic/setup.bash
$ cd ~/ws_moveit
$ git clone
https://github.com/UniversalRobots/Universal_Robots_Client_Library
.git src/Universal_Robots_Client_Library
$ sudo apt update -qq
$ sudo apt install python3-vcstool
$ vcs import --input
src/Universal_Robots_ROS_Driver/.noetic.rosinstall src
$ git clone
https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.git
src/Universal_Robots_ROS_Driver
$ rosdep update
$ rosdep install --from-paths src --ignore-src -y
$ catkin_make_isolated
$ source devel_isolated/setup.bash
```

- Universal Robots MoveIt configuration package: https://github.com/ros-industrial/universal_robot

```bash
$ cd ~/ws_moveit/src
$ git clone -b noetic-devel https://github.com/ros-
industrial/universal_robot.git
$ cd ~/ws_moveit
$ rosdep update
$ rosdep install --rosdistro noetic --ignore-src --from-paths src
```

- Intel Realsense Description: https://github.com/IntelRealSense/realsense-ros/tree/ros1-legacy

```bash
$ sudo apt-get install ros-$ROS_DISTRO-realsense2-camera
$ sudo apt-get install ros-$ROS_DISTRO-realsense2-description
```

- ArUco ROS: https://github.com/pal-robotics/aruco_ros

```bash
$ cd ~/ws_moveit/src
$ git clone https://github.com/pal-robotics/aruco_ros.git
$ rosdep install --from-paths src --ignore-src -r -y
```

- Install PYQT5 for GUI:

```bash
$ sudo apt install python3-pyqt5
```

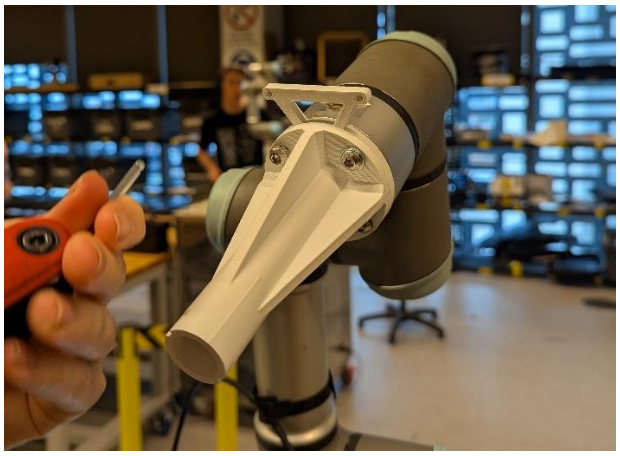
3. Installation Guide

3.1. Hardware Installation

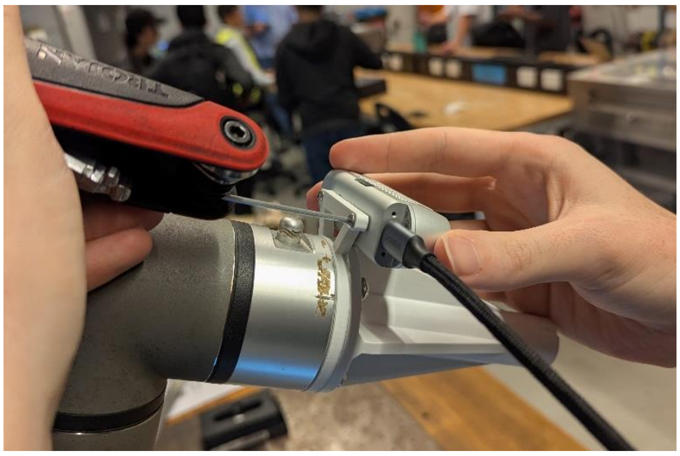
1. Clear workspace around UR3 robot. Set up safety barriers around workspace if necessary and eliminate any hazards to operation (tripping, electrical, etc.).

2. Connect your computer with the UR3 using the ethernet cable. This should connect to the router interfaced with the UR3.

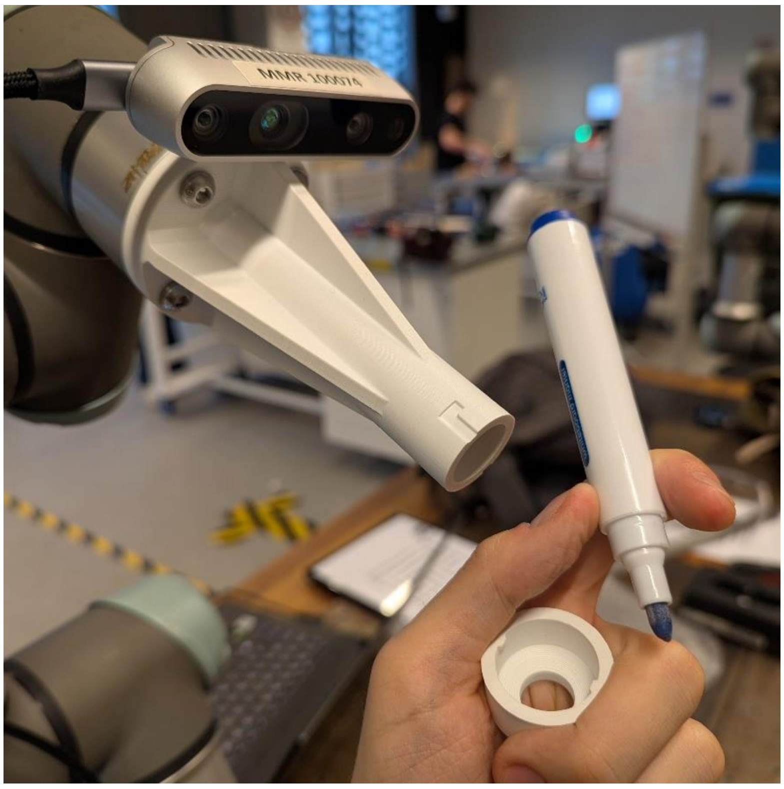
3. Using the four M6 bolts, fix the Custom RealSense camera mount and marker holding end effector to the mounting points on the end of the UR3 in the illustrated order and orientation:



4. Attach the Intel RealSense camera to the camera mount using the two M3 bolts and plug the camera into the computer using the USB-C to USB-3.0 cable.



5. Open the cap to the custom UR3 end effector and insert an erasable marker, closing the cap back up afterwards to ensure a secure fit.



3.2. Software Installation

1. Clone the MazeRunner GitHub repository:

```bash
$ cd ~/ws_moveit/src
$ git clone https://github.com/HughRad/MazeRunner.git
```

2. Install dependencies:

```bash
$ cd ~/ws_moveit/src
$ sudo apt-get update
$ rosdep install --from-paths src --ignore-src -r -y
```

3. Build and source the catkin workspace:

```bash
$ cd ~/ws_moveit
$ catkin build
$ source ~/ws_moveit/devel/setup.bash
```

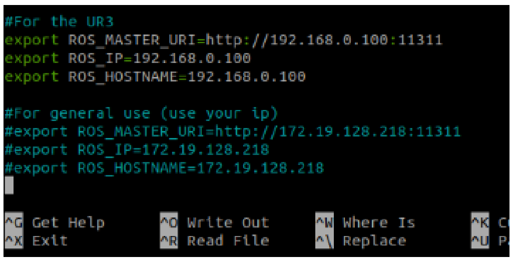
4. Operation Guide

Once all required hardware and software has been installed/set up, review the following steps to operate maze runner:

1. Power on the UR3 and keep the attached UR3 teach pendant in reach.

2. Setup IP configuration for connecting to Robot:

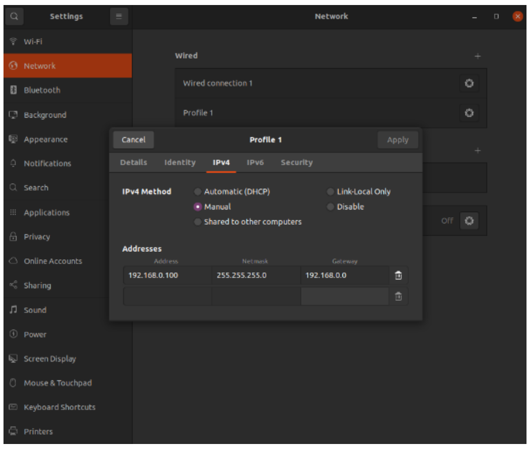
```bash
	$ nano ~/.bashrc
	Make relevent edits/additions as per the image below
	Ctrl X to exit (follow prompts)
	$ source ~/.bashrc
```



```
#For the UR3
export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_IP=192.168.0.100
export ROS_HOSTNAME=192.168.0.100

#For general use (use your ip)
#export ROS_MASTER_URI=http://172.19.128.218:11311
#export ROS_IP=172.19.128.218
#export ROS_HOSTNAME=172.19.128.218

^G Get Help     ^O Write Out    ^W Where Is     ^K C
^X Exit         ^R Read File    ^\ Replace      ^U P
```



Profile 1 — IPv4

IPv4 Method: Automatic (DHCP) [✓ Manual] Shared to other computers Link-Local Only Disable

Addresses

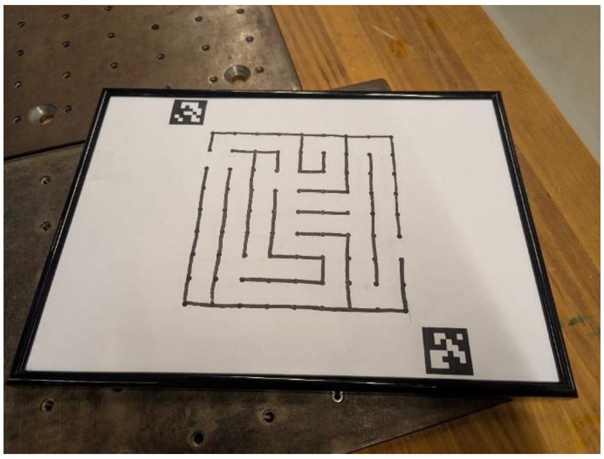
Address	Netmask	Gateway
192.168.0.100	255.255.255.0	192.168.0.0

The IP address is specific to your system, check the network requirements for your device on the UR pendant and create your profile accordingly.

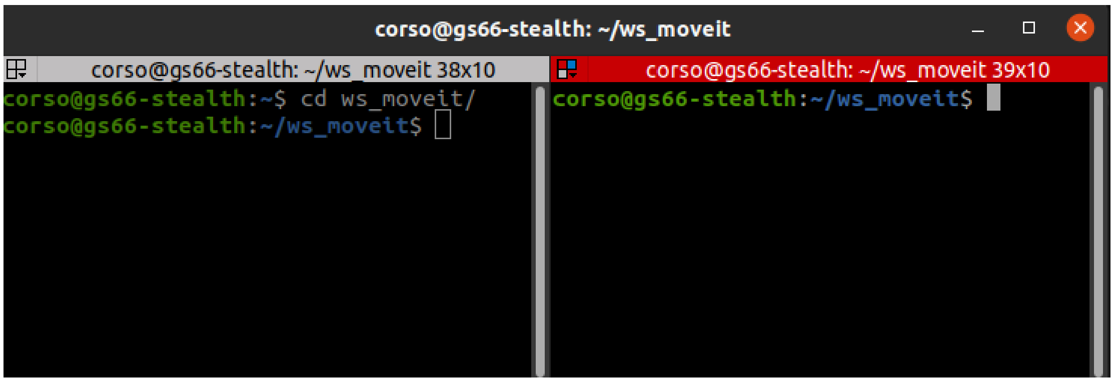
3. Build and source the catkin workspace:

```bash
$ cd ~/ws_moveit
$ catkin build
$ source ~/ws_moveit/devel/setup.bash
```

4. Draw a maze on the maze grid template and place it in front of the UR3. Orientation does not matter; however, it must be within the real sense cameras search area workspace. This may require you to dry run the system at least once while watching the camera feed to determine the exact boundaries of this area.



5. Open a copy of the Terminal such that you have two in the workspace directory:



```
corso@gs66-stealth: ~/ws_moveit
corso@gs66-stealth: ~/ws_moveit 38x10          corso@gs66-stealth: ~/ws_moveit 39x10
corso@gs66-stealth:~$ cd ws_moveit/            corso@gs66-stealth:~/ws_moveit$
corso@gs66-stealth:~/ws_moveit$
```

6. In one terminal run:

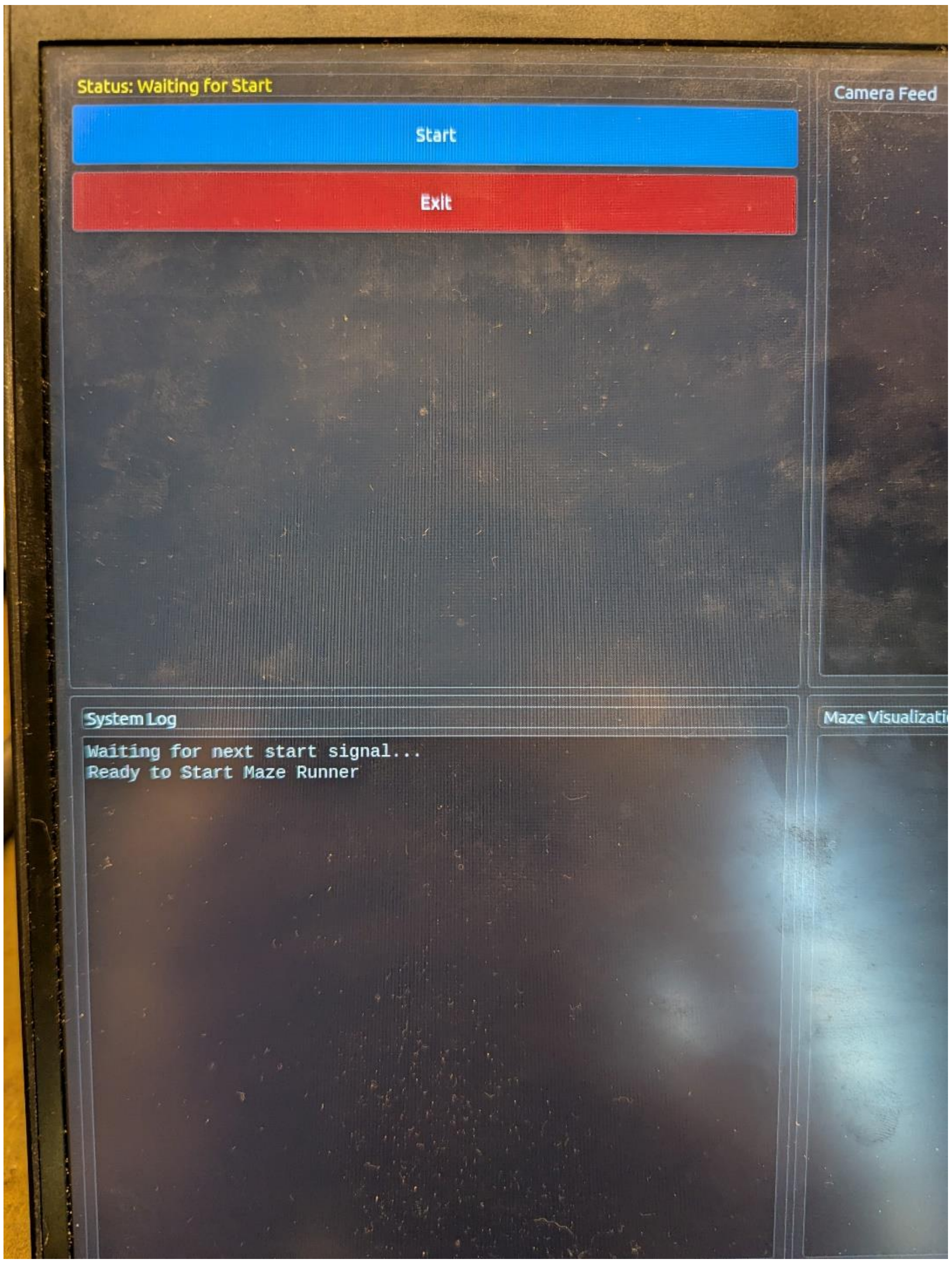
```bash
$ roslaunch ur_robot_driver ur3_bringup.launch robot_ip:= <your
robot ip>
```

The robot IP is unique to the robot in use and should be obtained from the UR pendant.

7. In the other terminal, run:

```bash
$ roslaunch maze_runner maze_runner.launch
```

8. In the GUI that will open after running the above command, wait for the robot to move to the starting posision and for the system log to report “Ready to start Maze Runner”, after which select “Start” to begin robot operation.



```
Waiting for next start signal...
Ready to Start Maze Runner
```

5. Subsystem Breakdown

Maze Localisation & Capture:

- **Purpose**: Captures a maze image using velocity-based control, guided by ArUco markers and live camera feed.
- **Key Topics/Services**:
    - /camera/image_raw → Real-time camera feed from the RealSense camera
    - OpenCV ArUco Detector → Marker detection for alignment
    - End-effector movement control
- **Inputs**:
    - Camera feed (RGB)
    - ArUco marker positions
- **Outputs**:
    - image.jpg (maze image for processing)
    - Velocity and rotation data

Image Processing & Maze Detection:

- **Purpose**: Converts the captured maze image into a structured ASCII array representation after boundary detection and noise removal.
- **Key Topics/Services**:
    - OpenCV ArUco Detector → Detects maze boundaries
    - Hough Line Transform → Detects wall lines
    - Thresholding & Contour Detection → Extracts maze paths
- **Inputs**:
    - image.jpg (from Maze Localisation & Capture)
- **Outputs**:
    - ASCII string representation of the maze ('#' for walls, '.' for paths)
    - Visualisation images for debugging

Maze Solving & Path Planning Algorithm:

- **Purpose:** Solves the grid-based maze and generates a list of cartesian waypoints for robot navigation.
- **Key Topics/Services:**
    - Breadth-First-Search (BFS) → pathfinding algorithm
- **Inputs:**
    - ASCII maze string array (from Image Processing & Maze Detection)
    - World coordinate, scaling, rotation and drawing depth modifiers
- **Outputs:**
    - List of cartesian coordinates for robot path execution
    - ASCII representation of the maze solution for debugging

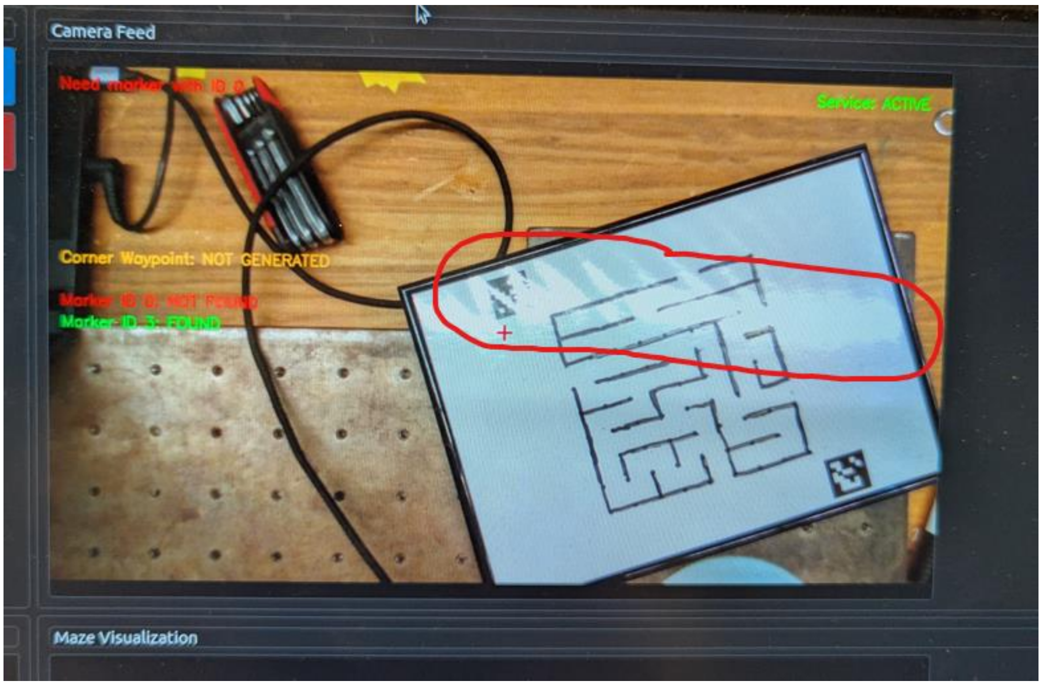
Robot Motion & Control:

- **Purpose**: Integrates image processing with motion planning to plan and execute a cartesian based trajectory via waypoints as determined by the integrated systems.
- **Key Topics/Services**:
    - Robot joint states → to determine robot position/orientation

  - Waypoint data, image feed, and snapshot data from localisation → to enable the callbacks and subsequent functionality
  - Service to call for localisation node start up
- **Inputs**:
  - List of cartesian coordinates (Poses) → obtained from Path Planning Algorithm
  - Image based waypoint and rotation data → from maze localisation subscribers
  - Camera Image → from maze localisation
- **Outputs**:
  - Computed and performed Robot trajectory tracing maze path

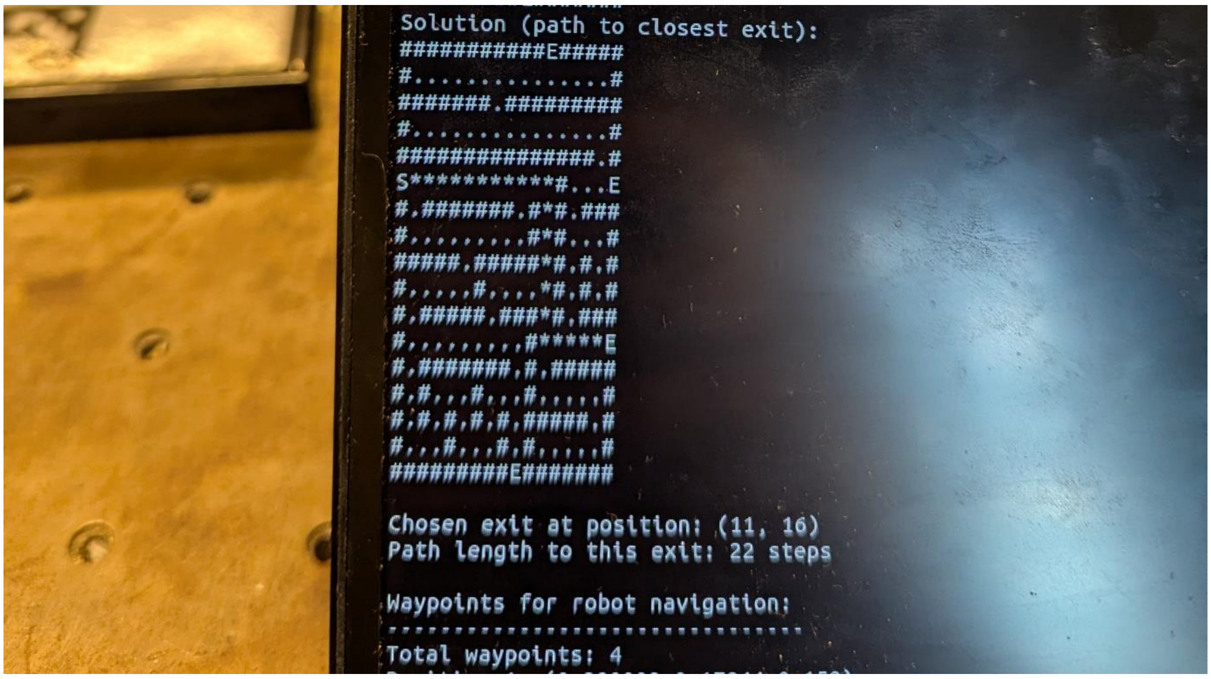
6. Troubleshooting & FAQs

- **I have run the robot running program; however, it fails to read the maze placed in front of it and returns to the starting position.**
  - The Maze grid/Aruco markers may not be placed fully in the cameras field of view/workspace, run the program again and verify it is by viewing the camera output. Move the maze if necessary.
  - The maze/Aruco markers may be obstructed from the camera by glare from nearby lights. Consider turning off/obstructing any lights directly above or next to the maze.



```
Need marker with ID 0
Service: ACTIVE
Corner Waypoint: NOT GENERATED
Marker ID 0: NOT FOUND
Marker ID 3: FOUND
```

- **The robot seems to detect the maze fine, however, fails to solve it even though it clearly has a solution.**
  - Check the running terminal when in operation to see what ASCII maze representation it outputs. Check to see if this corresponds to the drawn one. If not, glare or some kind of other obstruction may be interfering with the cameras view. Otherwise, the maze simply may not be drawn neat enough for the camera to detect the maze properly.

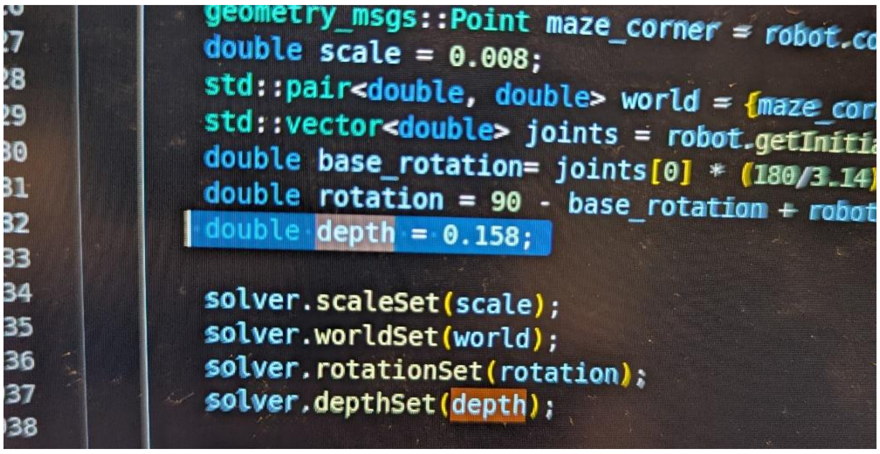


```
Solution (path to closest exit):
###########E#####
#...............#
#######.#########
#...............#
###############.#
S***********#...E
#.#######.#*#.###
#.........#*#...#
#####.#####*#.#.#
#.....#....*#.#.#
#.#####.###*#.###
#.........#*****E
#.#######.#.#####
#.#...#...#.....#
#.#.#.#.#.#####.#
#...#...#.#.....#
#########E#######

Chosen exit at position: (11, 16)
Path length to this exit: 22 steps

Waypoints for robot navigation:
------------------------------
Total waypoints: 4
```

- **The robot seems to solve the maze fine; however, the marker does not press down far enough on the maze to draw the solution or pushes down excessively hard.**
  - The supplied drawing depth may need adjustment if your workspace if a different height. This value can be adjusted in the “main” function of the “robot_control” class. The value is named “depth” and by default is set to 0.158 meters.



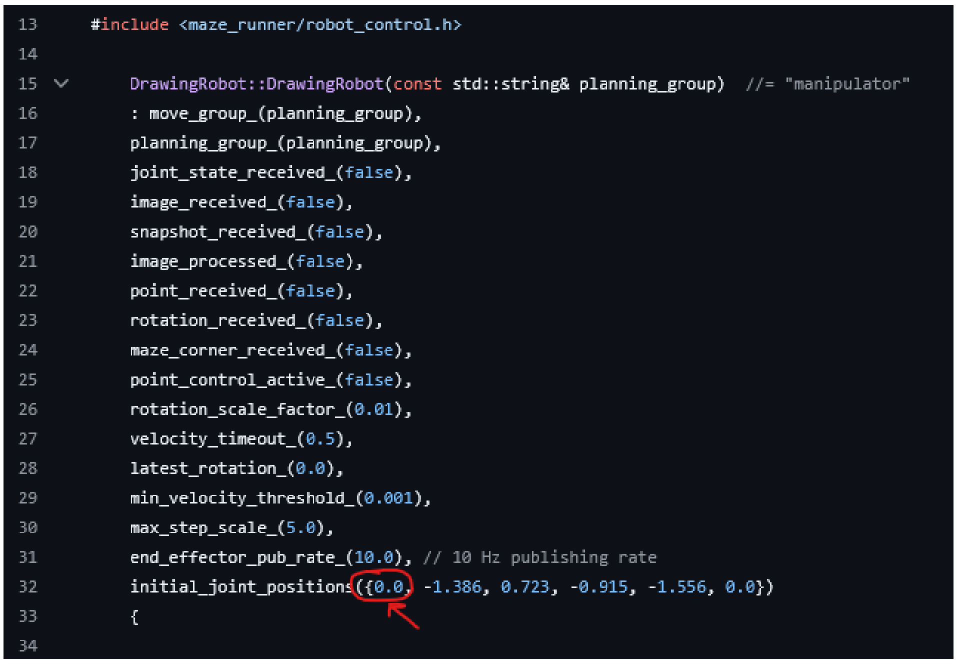
```
    geometry_msgs::Point maze_corner = robot.co
    double scale = 0.008;
    std::pair<double, double> world = {maze_cor
    std::vector<double> joints = robot.getIniti
    double base_rotation= joints[0] * (180/3.14)
    double rotation = 90 - base_rotation + robot
    double depth = 0.158;

    solver.scaleSet(scale);
    solver.worldSet(world);
    solver.rotationSet(rotation);
    solver.depthSet(depth);
```

- **The robot did not use the point in the maze I intended to use as the start as the entrance, instead using it as the exit or not using it at all.**
    - Maze runner has no function to choose which entrances will be start points or end points. The system will search each side of the maze one at a time and use the first entrance it sees as the start, assigning all other entrances it finds as exits. It will then path plan from the assigned start point to whichever exit is closest.

- **I have done all the setup correctly, but my code still refuses to interface with the UR3.**
    - Double check that the robot you are using is a UR3, our system is not configured by default to use a UR3e which look identical.

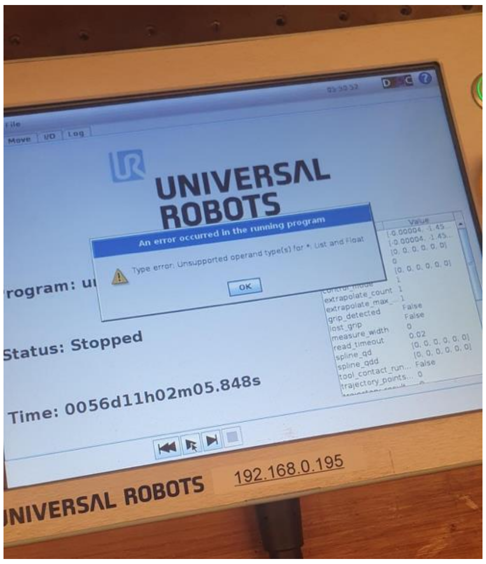
- **The robot faces the incorrect position during operation, resulting in it not scanning the desired workspace.**
    - Maze runner can operate towards the 'front, 'left', or 'right' of the robots mounting position. This can be changed in the "robot control" class by changing the "initial_joint_positions" value for joint one. By default, this is set to 0 radians (right position). This value must be set to either "0", "1.57" or "3.14".



```
#include <maze_runner/robot_control.h>

    DrawingRobot::DrawingRobot(const std::string& planning_group)  //= "manipulator"
    : move_group_(planning_group),
    planning_group_(planning_group),
    joint_state_received_(false),
    image_received_(false),
    snapshot_received_(false),
    image_processed_(false),
    point_received_(false),
    rotation_received_(false),
    maze_corner_received_(false),
    point_control_active_(false),
    rotation_scale_factor_(0.01),
    velocity_timeout_(0.5),
    latest_rotation_(0.0),
    min_velocity_threshold_(0.001),
    max_step_scale_(5.0),
    end_effector_pub_rate_(10.0), // 10 Hz publishing rate
    initial_joint_positions({0.0, -1.386, 0.723, -0.915, -1.556, 0.0})
    {
```

- **I am seeing an error appearing on the UR pendant that stops the program, why does this happen?**
  - If it is the error below, we suspect this a result of the ROS drivers reaching their EOL resulting in deprecation, however that has not yet been confirmed and thus this has no robust solution.
  - If you can press ok on the message and restart the drivers with the robot with play before it receives any commands the system should continue, however, should this occur mid command it will result in an irreconcilable error and the program must be restarted.



```
An error occurred in the running program
Type error: Unsupported operand type(s) for *: List and Float
OK
```

```
Program: u
Status: Stopped
Time: 0056d11h02m05.848s
```

```
                        Value
                        [-0.00004, -1.45...
                        [-0.00004, -1.45...
                        [0, 0, 0, 0, 0, 0]
                        0
                        [0, 0, 0, 0, 0, 0]
                        1
control_mode            1
extrapolate_count       1
extrapolate_max_...     1
grip_detected           False
lost_grip               False
measure_width           0
read_timeout            0.02
spline_qd               [0, 0, 0, 0, 0, 0]
spline_qdd              [0, 0, 0, 0, 0, 0]
tool_contact_run...     False
trajectory_points...    0
```