**AUTO4508 Mobile Robots**

**Pioneer 3-AT Autonomous Navigation**

**Group 8**

# Design Overview:

The design was implemented using the ROS framework and attempted to achieve map-less autonomous navigation to GPS waypoints (with obstacle avoidance). This design was made under a package titled ‘group8’.

Equipment:

* Pioneer 3-AT Outdoor Mobile Robot Platform
* Industrial Linux PC with touch screen display
* GPS: built-in
* IMU: Phidget Spatial 3/3/3
* Camera: Stereo Camera OAK-D V2
* Controller: DualShock III clone

Requirements and Dependencies:

The design used ROS Noetic, installed on ubuntu, and has the following ROS package dependencies:

* joy
* teleop\_twist\_joy
* rosaria
* nmea\_navsat\_driver
* tf
* phidgets\_spatial
* gps\_common
* robot\_pose\_ekf
* depthai\_ros
* depthai\_examples
* depthimage\_to\_laserscan
* move\_base
* navigation
* rviz

It is recommended that the full ROS noetic package is installed to avoid missing packages.

The utm python package is also required.

# Gamepad control implementation:

The gamepad control was implemented using the joy, telelop\_twist\_joy and rosaria nodes. The joy node reads inputs from the gamepad and publishes a joy topic that describes the state of the gamepad. This topic is subscribed to by teleop\_twist\_joy to produce a command velocity based on the position of the left joystick. This topic will however only publish when the L1 button is held down. This L1 button also acts as a deadswitch for the autonomous navigation, causing the robot to stop moving when it is held down as empty velocity commands are passed to the robot. This velocity command published by teleop\_twist\_joy is subscribed to by rosaria, which controls the robot’s motors causing it to move in the desired direction. Depending on the robot used, this control will also allow for reverse movement. Robots 3 and 5 were found to have this functionality.

Below is the node connection that allows for gamepad control:

# A picture containing text Description automatically generatedAutonomous navigation implementation:

Diagram

Description automatically generated

Figure 1: Navigation Stack Setup [1]

The navigation stack for the robot was set up following the tutorials from the ROS wiki, which required tf, odometry and sensor messages to be passed to the move\_base.

## Odometry:

The odometry input to move\_base was created as a combination of the available GPS, IMU and odometry data using robot\_pose\_ekf.

The GPS data was obtained from the GPS sensor and the nmea\_navsat\_driver package, which outputs a NavSatFix message from the sensor.

The IMU data was obtained from the IMU sensor and the phidgets\_spatial package, specifically the PhidgetsSpatialNodelet in that package. This outputs an imu topic to ROS.

The odometry data was obtained from the odom topic published by rosaria.

Before these inputs could be passed to robot\_pose\_ekf, some conversion and adjustment of the data were required.

robot\_pose\_ekf requires the GPS input to be an odometry message type, so the gps\_common node was used to convert the NavSatFix gps message to the required odometry message. This conversion would convert the longitude, latitude and altitude measurements into x, y, z UTM coordinates.

It was found that the raw IMU data did not publish any information about the orientation of the robot, so the first value of the orientation covariance was set to -1 (as recommended by the documentation for /sensor\_msgs/imu) and the orientation of x set to 1 to resolve the ‘quarternion does not resolve to 1’ error thrown by robot\_pose\_ekf. These changes were made by writing a node titled ‘imu.py’ to subscribe to the imu topic and publish the corrected data.

The odometry data from rosaria was missing the covariance for its pose and twist, so a node titled ‘covariance.py’ was written to give small covariance values to the pose and twist to resolve an error thrown by robot\_pose\_ekf.

With these changes, a PoseWithCovarianceStamped message would be published by robot\_pose\_ekf, which would need to be converted to an Odometry message type to be accepted by move\_base. This conversion was done by a custom node titled ‘odom\_fix.py’. The output from robot\_pose\_ekf would combine the inputs from each of the 3 sensors to give the best estimate of the robot’s current position and orientation. The pose of the robot is given in absolute UTM coordinates.

A picture containing text

Description automatically generated

## Sensor sources:

The sensor source is from the camera, and the outputs from the camera are obtained by launching the stereo\_node from depthai\_examples. This node gives a depthimage output, which is used to generate a laserscan with the node depthimage\_to\_laserscan. This is passed directly to move\_base.

A picture containing chart

Description automatically generated

## Move base:

The move\_base navigation implementation was adapted from ‘p3at\_tutorial’ made by Gastd, available from <https://github.com/Gastd/p3at_tutorial>. The mapless demo included in this tutorial is used. This move\_base does not use a map, and instead gives the global\_costmap the robot\_pose\_ekf/odom\_combined frame. The overall node configuration of the implementation is as shown below:

Diagram

Description automatically generated

Once a goal is published to move\_base, it will publish a command velocity to rosaria to move the robot to the goal.

TF:

The tf tree setup is as shown below: Diagram

Description automatically generated

The robot\_pose\_ekf/odom\_combined frame is in absolute UTM coordinates, while all the other sensors connect to the base frame, defined as a point on the robot.

# Waypoint navigation:

A custom node using the SimpleActionClient would send GPS waypoints to the move\_base. These waypoints would be converted into UTM coordinates to be consistent with the position of the robot provided by robot\_pose\_ekf. This node would send a goal, then wait for a response from move\_base to say that the goal has been reached before sending the next waypoint.

# Launching the software:

There are 4 separate launch files used in this implementation:

* group8.launch: launches all the sensor nodes and required tf transforms. Run using:

roslaunch group8 group8.launch

* move\_base\_mapless\_demo.launch: launches the move\_base and the costmaps. Run using:

roslaunch group8 move\_base\_mapless\_demo.launch

* rviz.launch: launches rviz to display the robot’s path and costmaps. Run using:

roslaunch group8 rviz.launch

* move.launch: launches a node that publishes gps waypoints to move\_base. Run using:

roslaunch group8 move.launch

## Limitations:

Testing of the robot’s functionalities were not all successful. The gamepad control was tested to be working correctly on multiple robots, but the autonomous navigation system was unsuccessful on all 4 of the robots we tested:

* Robot 1 would not output any GPS data, so robot\_pose\_ekf would not publish any odometry
* Robot 2 had GPS working but did not have a working camera
* Robot 5 also would not output any GPS data
* Robot 4 initially appeared to have both the camera and GPS working but would not drive to the correct waypoints. Towards the end of the testing, the robot began to freeze and crash after only launching the sensor and move\_base nodes. This robot also could not drive backwards.

We were unable to test robot 3 as it had been booked by another group for almost the entirety of the project duration.

The image processing portions of the project were unable to be implemented due to time constraints and the failure of the navigation system.

## Group contributions:

|  |  |
| --- | --- |
| ROS gamepad implementation | YiMing |
| ROS navstack setup | YiMing, Joris, Jialing |
| Software installation/repair on robots | YiMing, Joris |
| Video | Joris, Jialing, Kate |
| Project Design Report | YiMing, Kate |
| User Manual | Joris, YiMing |

References

[1] "navigation/Tutorials/RobotSetup - ROS Wiki", Wiki.ros.org, 2022. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Accessed: 26- May- 2022].