MINION

custom built Robot at AARG Lab

International Institute of Information Technology
Hyderabad - 500 032, INDIA

1 Handling Instructions

- Take utmost care while handling the robots as the robot is heavy. Do not carry the robot upside down, as it may cause serious damage to the robot.
- Power ON/OFF switch is given to switch the robot ON/OFF. Do not power off the robot using the ON/OFF switch without shutting down the Raspberry Pi.
- Make sure the robot has sufficient battery before using it for any purpose. A robot should maintain a minimum of 10.5 volts all the time. Charge it fully and keep monitoring the battery while in use. It may cause a serious damage to the battery if used on low battery charge.
- The batteries do not have a over charging protection. Make sure that you unplug the robot when battery reaches 12 volts. (NOTE: Minion usually takes 2 hours to get fully charged, i.e. from 10.5 to 12 volts). So, keep track of the time.
- Do not keep a payload heavier than 5 kgs on each robot.
- Do not open the lid of the robot unless necessary. Only remove the robot's lid if you need to make a wiring correction or any addition in the hardware.
- Do not try to stop the powered motor of the robot when they are running on battery. This might affect the motor performance and can cause serious damage to the battery of the robot.

2 Prerequisites

- Knowledge of Python Programming
- Basic Knowledge of Robotic Operating System (ROS)
- Knowledge of Basic Electronics and Hardware
- Arduino and Arduino Programming

Kindly go through the above topics before starting any experimentation on the robots.

3 Introduction

We build a custom robot at the AARG labs named $Minion$, which is capable of carrying heavy load (each robot can carry upto 5 Kgs). The robot uses robust DC geared high torque motors providing accurate wheel encoders readings. To carry the replacement operation, each robot has a linear actuator to move the payload up and down. We discuss the robot hardware in detail in this chapter. The chapter also discusses the modeling of various robot models like kinematic, dynamic, battery, friction which are important to control various aspect of the payload transport. In the end, we will highlight the formation control of multiple robots using a decentralized control architecture and also find out various ways of robot localization techniques.

4 Robot Hardware and capabilities

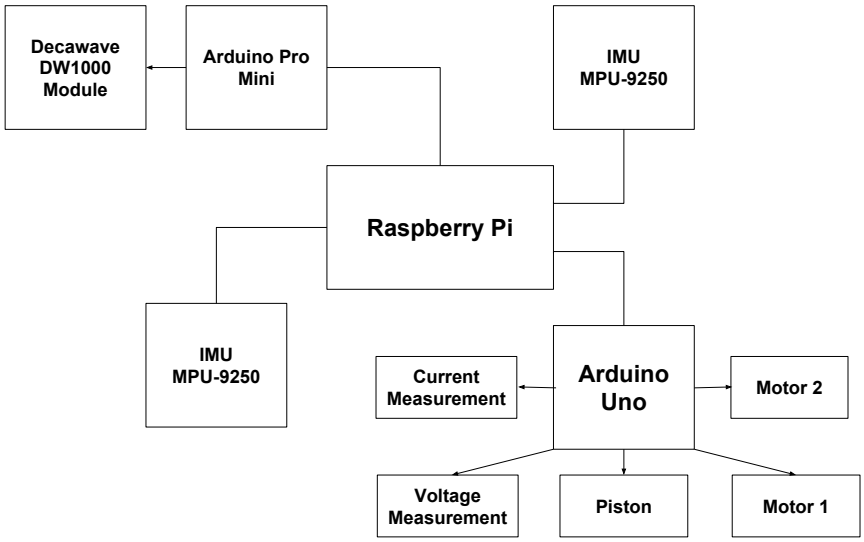


Figure 1: Block diagram of Minion

Now-a-days, there are various robots available in the market like Firebird [1], turtlebot [2] [3], Pioneer-3DX [4], Husky [5], etc. While these robots works well in their own ways, but when it comes to payload transport, the requirement is different in terms of high torque, heavy weight carrying capacity, accurate positioning and different localization methods. We will discuss these requirement in this chapter. A block diagram of the robot highlighting the building blocks of the robot is shown in Fig. 1 and circuit diagram of the robot with all the wiring and pin-out is displayed in Fig. 2.

Each robot comprises of three main controller boards: Raspberry Pi, Arduino Uno

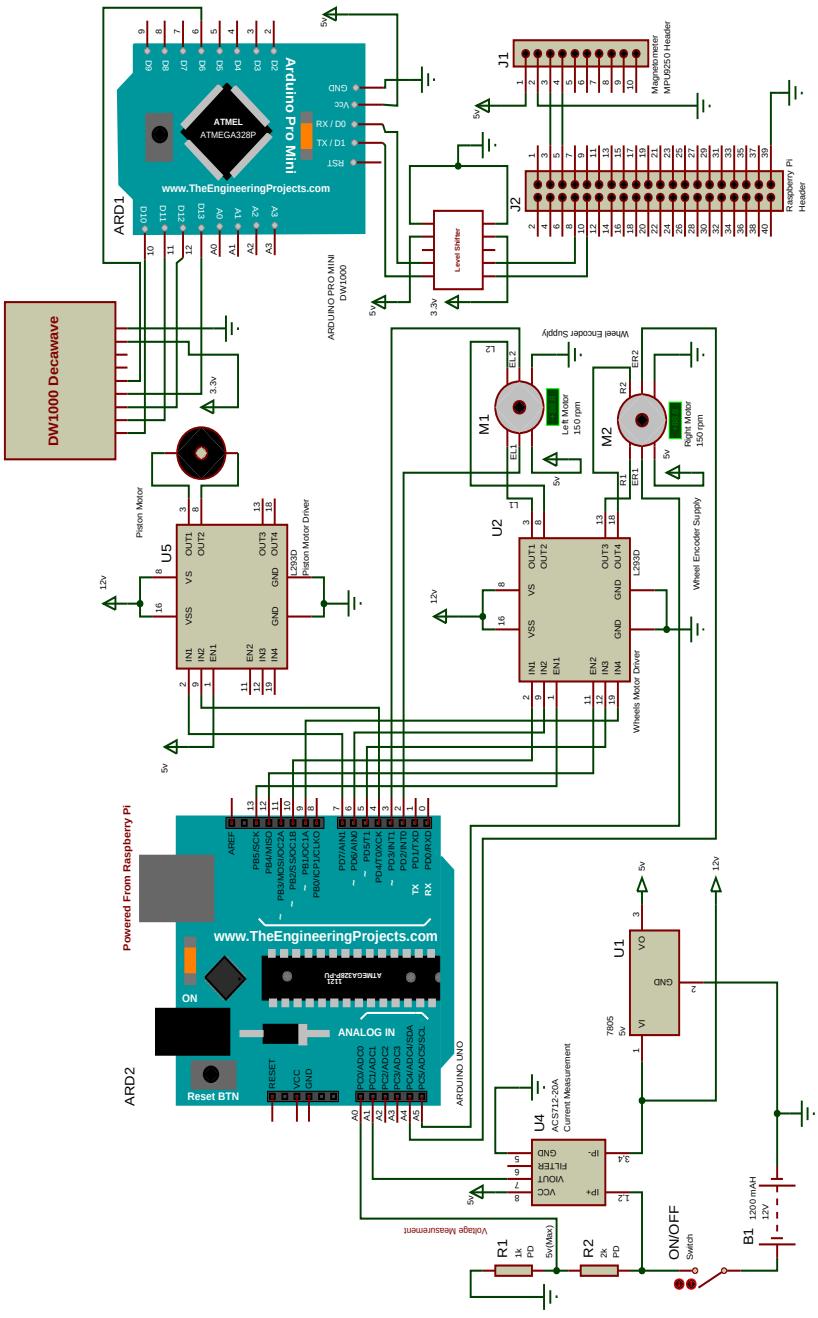


Figure 2: Minion internal design built at AARG Lab, IIIT Hyderabad.

and Arduino Pro mini. Raspberry Pi is used to control the high level tasks in the robot and is the master board which supervises the arduino uno and the arduino pro mini. It runs ROS (Robotics operating system) stack on Ubuntu Mate kernel. All the communication within the robots is taken care of by the ROS stack. Raspberry pi also control the MPU9250 9-axis sensor to get the accurate orientation of the robot and is connected using the I2C (Inter Integrated Communication) interface. The programming language used throughout for simulation as well as on hardware is Python.

Arduino Uno is a controller board is responsible for low level controlling operation within the robot such as moving the motors by applying PID (Proportional-Integral-Derivative) control on the desired and actual control input velocities, controlling piston up and down motion, measuring the current and voltage of the battery. The programming language used is C. The UNO board is USB powered from the raspberry Pi and receives the command to execute serially.

Arduino Pro mini is also used to connect the DW1000 UWB (Ultra Wide Band) module (Decawave). DW1000 is a ultra wide band frequency range based transceiver, which send and receive signal from the other modules to find the distance of the robot wrt the global cordinate system. The mini board sends the distances to the raspberry pi, which further used the data to find the accurate position of the robot. As the Pi board works on 3.3v, we shift the 5v signal from the mini board to 3.3v for the raspberry pi and vice-versa. We will discuss the concept of DW1000 module and localization of robot in detail in the latter part of this chapter.

The robot is a differential drive robot which has one castor wheel in the front and two 150 rpm DC geared Pololu motors which has internal hall effect based wheel encoders to provide accurate wheel odometry. Each motor gives 4488 counts per revolution which gives an added advantage in calculating the position of the robot.

Robots and Cost	
Robot Name	Cost (in INR)
Firebird V	$35,000$
Turtlebot 3 Burger	$39,000$
Turtlebot 2 Base kit	$1,00,000$
Turtlebot 3 Waffle	$1,00,000$
Pioneer P3DX	$2,85,500$
Minion	< 25000

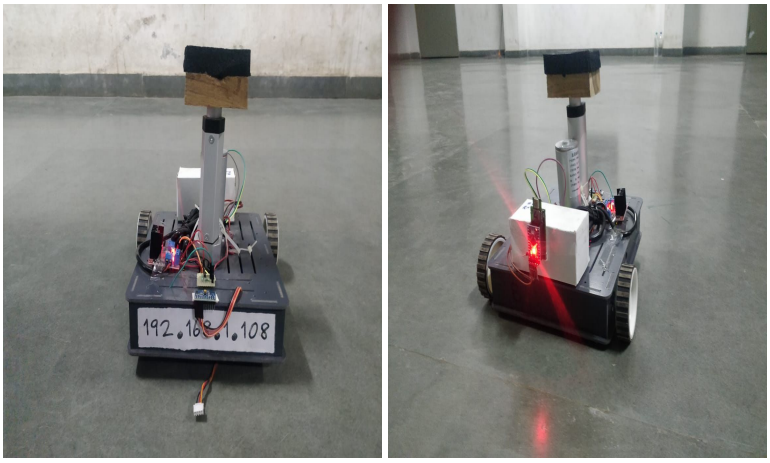
Table 1: Cost comparison of Minion with other standard robots available

A linear actuator or Piston is also present on the top of each robot which is used to carry out robot replacement in case of a low battery failure. The actuator used is a 4 inch stroke actuator i.e. it can move the load up by 4 inches. Whenever a robot has a low battery and needs a replacement, all the robots except the failed one moves the piston up and replacement takes place. The replacement mechanism is also discussed in detail towards the end of this thesis.

Each robot has a 1200mAH, 12volt Lithium Ion (Li-ion) battery which runs the complete system. The battery is enough to run each robot for about 1hr to 2hour, depending upon the load, the system is carrying. A voltage and current measurement

circuit is also placed on each robot to provide precise voltage and current values at any given point of time which gives an estimate of the remaining battery life of the robot. All the robots are wifi and ROS enabled. A robot is given a static IP address that allows the robots to communicate within themselves to share the data to perform any task.

The total cost of the robot including all the expenses is around $21,000$ INR which is less as compared to the available robots in the market and with much more capabilities for the purpose of payload transportation. A cost comparison is made in Table 1 with some of the available robot in the market and the Minion (Our custom built robot). Fig. 3 shows the final hardware design.



(a) Front view of Minion (b) Rear view of Minion

Figure 3: Final Hardware design of the Minion.

5 Components Used

- Raspberry Pi 3B
- Arduino Uno
- Arduino Pro Mini
- DW1000-UWB module
- MPU9250-IMU
- 64 CPR 70:1 Gear Ratio Polulu Motors with wheel Encoders
- Linear Actuator
- L293DE Motor Drivers

- ACS712 Current Measurement Circuit
- Li-Ion 1200 maH Battery
- Level Converter (5v to 3.3v)
- 12v to USB power Converter
- One Castor wheels
- Two Rear wheels
- ON/OFF Button
- Jumper Wires
- One Chassis

6 Software Used

Each robot has a ROS support and wifi connectivity which makes it easier to work with the robots from anywhere. The complete software is available on the github repository and can be found on $Robot\ Code$. These are the following requirement you need to fulfill to run the robot from your Laptop/PC.

6.1 System Requirement

- Intel i5 or above
- 4GB Memory or above
- At least 7 GB of free disk space
- Ethernet port or WiFi Support

6.2 Software Dependencies

- Install Ubuntu 14.04 or higher
- Python 2.6 or higher
- ROS (Install Kinetic or higher). Click $here$ to install.

6.3 ROS Packages Dependencies

- $Robot\ Localization$
- $IMU\ Complementary\ Filter$
- $ROS\ Arduino\ Bridge$

6.4 Other Dependencies

- $IMU\ Library\ (MPU9250)$

7 Connection with the Robot

Each robot is given an static IP which can be accessed by connecting to the AARG Wifi Network. The network is specifically dedicated to work with the robots.

- Connect your Laptop/PC to "AARG" wifi network.
  - Username: **AARG**
  - Password: **aargnet123**

  NOTE: Do not share the password of this network with others, as this may cause lag while sending commands to the robot.

- Open a terminal in your Laptop. (Use **Ctrl+Alt+T**)
- SSH into the robot using the below credentials
  - Username: **pibot**
  - Password: **pibot**
    For Example: Type **ssh -X pibot@ip**. Robot IP is present at the front of the robot.

You have successfully logged into the robot.

7.1 Step 1 (Laptop/Master)

We suggest that you should make your Laptop/PC as master and the robots as slaves to ease the process of subscribing the topics. This will allow you not to log-in into a robot in a different terminal for each topic you want to subscribe. Follow the commands below to make your system as a master.

- Open a terminal in your laptop and get your IP address (Master IP). Type **ifconfig** for Ubuntu 16.04 or below and **hostname -I** for above
- Type **export ROS_MASTER_URI=http://master-ip:11311**
- Type **export ROS_IP=master-ip**

For Example: If your master IP is **192.168.1.200**, then

- Type **export ROS_MASTER_URI=http://192.168.1.200:11311**
- Type **export ROS_IP=192.168.1.200**

7.2 Step 2 (Robot/Slave)

- Type **export ROS_MASTER_URI=http://master-ip:11311**
- Type **export ROS_IP=robot-ip**

For Example: If your master IP is **192.168.1.200** and robot IP is **192.168.1.101**, then

- Type **export ROS_MASTER_URI=http://192.168.1.200:11311**
- Type **export ROS_IP=192.168.1.101**

Do the steps for all the robots you want to run at a time. This will enable the robots to publish/subscribe the topics without log-in to the robot again.

8 Using the Robot Codes

To make the robot execute a functionality, there are few codes which should be referred. These codes along with their packages are listed below.

- **Ros_arduino_bridge**: This package includes the low level control architecture of the robot. Arduino Uno is used to control the motors of the robot. Raspberry Pi acts as a master and Uno as a slave. We send commands from Pi to Uno for getting the encoder counts, setting the PID values, accessing the baud rate, etc. All these parameter can be tuned using this package.
  - **Ros_arduino_python**
    - **Config**
      - **my_arduino_params.yaml**: It include several tuning parameters such as PID values, baud rate, serial port number,robot specification, etc. We can change these parameters as per the usage of the robot. If the existing robot wheel has to be replaced, we need to change the parameter/dimensions of the new wheel in this file. PID parameters can be tuned if the robot low level control is not accurate. It can be checked by running a robot in a straight line. If robot moves in a perfect straight line, PID parameters are tuned perfectly. Baud Rate and Serial port values are hardware dependant. Check the port number at which the arduino is connected to the Raspberry Pi. Robot Parameters are specific to a particular robot (Robot Dimension, Wheel diameter, encoder counts etc).
    - **src/ros_arduino_python**
      - **arduino_driver.py**: The driver is used to create a communication channel between the Raspberry Pi and an Arduino Uno. Serial communication is used for this purpose. The serial channel can be changed in the **my_arduino_params.yaml** config file.
      - **base_controller.py**: The controller takes the paramters from the **my_arduino_params.yaml** file and the data received from the arduino (wheel encoder counts, voltage, current information) to calculate the odometry information of the robot. It publish topics *odom* of type $Odometry$, *voltage*, *current* of type $Float64$.
      - **Subscriber:** *cmd_vel*, *pistonListing*
      - **Input Parameter:** All parameters in **my_arduino_params.yaml**
      - **Publisher:** *odom*, *voltage*, *current*
- **Pibot** The package controls the high level functionality of the robot. The directory includes two sub-directories (Scripts and Launch). Scripts has all the python file in it which are responsible for implementing the high level control and launch files are a medium to run those files. We can also run the python files from the directory itself either by using *rosrun* or by *python* command.
  - **Scripts**

* **target_definition.py**: The file is used to define the command velocities of a robot. *vel* is the linear velocity of the robot and *omega* is the angular velocity of the robot. It publish the command velocity of the robot on a topic named *cmd_vel* in a $Twist$ format.
  - **Input Parameter:** *vel*, *omega*
  - **Publisher:** *cmd_vel*

* **get_distances.py**: The file is used to get distances from the decawave module. Arduino Pro Mini is used to collect the data from each robot and the anchors present on the wall. The distances are then sent to Raspberri Pi through Serial Pins (UART0: Pin 14 and Pin 15) for further processing. It publishes the distances on a topic named as *distances*.
  - **Publisher:** *distances*

* **trilateration_opti.py**: The distances received using the *get_distance.py* script are now used to calculate the exact position of the robot. We use a trilateration method that uses the distance received and applies an optimization to identify the location of the robot. The location is published in $Odometry$ message type with a topic named *d_odom*.
  - **Subscriber:** *distances*
  - **Input Parameter:** *max_tags*, *tag_locations*, *odom_frame_id*, *child_frame_id*
  - **Publisher:** *d_odom*

* **imu9250.py**: We can find the orientation of the robot using the wheel odometry, which is not a reliable source of finding the orientation of a robot. To get a global value of orientation, we use a separate sensor called IMU (Inertial Measurement Unit- which is MPU9250 in our case). It is an 9-axis sensor, comprising 3-axis Accelerometer, 3-axis Gyroscope, 3-axis Magnetometer. We use these values and apply a complementary filter on them to find the heading direction of a robot. The file publishes the accelerometer and gyroscope data in an IMU message format on a topic *imu/data_raw*. It also publishes the magnetometer data in a $MagneticField$ message type, on a topic named *imu/mag*.
  - **Publisher:** *imu/data_raw*, *imu/mag*

  See 10 to calibrate the magnetometer.

* **lf_bio.py**: This file include the control logic for a decentralized leader-follower control algorithm. The control system is implemented to make the follower robots follow the leader. It takes the leader information, the distance and angle of each robot to maintain from the leader and calculates the command velocities of the followers such that the

formation is maintained. It publishes the command velocities of the followers on a topic names *cmd_vel* on a $Twist$ message type.

- **Subscriber:** *leader/cmd_vel*, *leader/f_odom*, *follower/f_odom*
- **Input Parameter:** L_{ij}, Phi_{ij}, *k1*, *k2*, *k3*, *A*, *B*, *D*
- **Publisher:** *follower/cmd_vel*

Here, L_{ij},Phi_{ij} are the distance and angle respectively to maintain by the follower from the leader to remain in formation. k1, k2, k3, A, B, D are the formation gains.

* **stp_odom.py**: This is used to publish a static tf broadcaster between the world and wheel odometry. As we have two localization sources of the robot: wheel odometry which is the local source of localization and UWB localization which is a global source of localization. We link these two sources through this script. It does not subscribe or publish anything and only include a transformation from the odometry frame to world frame.

– **Launch** To run the above files in a less complex manner, we write a launch file where we can write all the files that need to be launched along with the parameters to the files. **If you are dealing with multiple robots, do not forget use separate name space of each of the robot.** The Description to some important files are given below.

* **ompl.launch**: The file is used to launch a minimal set of files required to run a robot in an environment. It includes launching python files to start the arduino for low level control, receiving distances from the decawave modules (anchors on the walls), running an optimization for trilateration (finding position of robot using decawave distances), imu imu for finding the robot orientation, generating the transformation between the world and odometry frame (wheel odometry), etc.
  To run the launch file open a terminal and write:
  roslaunch pibot ompl.launch

* **pibot.launch**: The file launch all the codes launch in *ompl.launch*. Along with those codes, it launch a file to run the robots in a formation. In the formation node, we also need to add the parameters to maintain a desired distance and angle from the leader.
  To run the launch file open a terminal and write:
  roslaunch pibot pibot.launch

* **leader_launcher.launch**: To run the leader with a command velocity, we use this file. Pass the arguments v_target, w_target to run the leader at a desired velocity.
  To run the launch file open a terminal and write:
  roslaunch pibot leader_launcher.launch

- **Other Codes**: Some additional codes needed to implement various functionalities on the robot are given below:

- **broadcast_piston.py**: Each robot has a piston/linear actuator mounted on them. To move the piston UP/DOWN we use this file. We need to provide a list of the robots and the status(UP/DOWN) of the piston. The id's of the robots in the list will execute the command on the piston based on the piston status. The list to be created is such that the first element is the status (1:MOVE_UP, 2:MOVE_DOWN, 0:STOP) and the rest of the elements are the robots id's. It publish a topic named *pistonListing* of message type $Int8MultiArray$. We can move the piston UP/DOWN within a limit of $0 - 4$ inches. Give the move UP command, followed by STOP command at any point of time to stop the piston to a particular height.
  - **Publisher:** *pistonListing*

- **getVoltages.py**: The file reads the voltage of all the available robots. It subscribes the topic name *voltage* from all the robots of message type $Float64$.
  - **Subscriber:** *voltage*

- **pibot_teleop.py**: To control the robot manually using a keyboard can be implemented using this script. The control will appear on the screen as the file is run. *w*, *a*, *d*, *s* are used to make forward, left, right, and backward movement of the robot. *x* is used to stop the robot. The robot accelerated with 0.05 m/sec with every press of *w* and decelerates with 0.05 m/sec by pressing *s*. The angular velocity increases by 0.1 rad/sec with *a* and decreases with *d*. It publish a topic *cmd_vel* for the robot to be controlled.
  - **Publisher:** *cmd_vel*

- **plot_heading.py**: We use this script to plot the heading/orientation of the robot. It subscribes to the topic *imu/rpy/filtered* of message type $Vector3Stamped$.
  - **Subscriber:** *imu/rpy/filtered*

- **plot_pose.py**: We use this script to plot the position of the robot. It subscribes to the topic *odom* of message type $Odometry$.
  - **Subscriber:** *odom*

- **generate_map.py**: We use this script to generate the map of the environment. It helps a robot to align itself to a leader and make a formation as per the set desired distance and angle in obstacle free environment. Open Motion Planning Library (OMPL) is used as a path planner and Pure pursuit controller is used as a path tracker. The file needs few arguments as mentioned below:
  - Leader ID
  - New Robot ID
  - List of all the robots

* Distance and angle of a robot to be replaced

The output of the file is a ***Pose.txt*** file, which will contain the position of the obstacles(robots other than the robot to move), start point and the goal point.

– **ompl_demo.py**: To run this file, ompl should be installed on the system (see details to install $ompl\_app\_python\_bindings$). The file uses the ***Pose.txt*** file generated by the *generate_map.py* file. Run the file to move a robot near the leader at a desired distance and angle.

* **Input File:** *Pose.txt*
* **Subscriber:** *newRobot/f_odom*
* **Publisher:** *newRobot/cmd_vel*

9 Robot Localization

To generate a global positional and orientation values, we run a $robot\_localization$ package. It takes the wheel odometry information, IMU values and decawave information to generate a global pose of a robot. Figure 4 shows the illustration of the messages distribution of different localization sources and generating a global pose. To run the

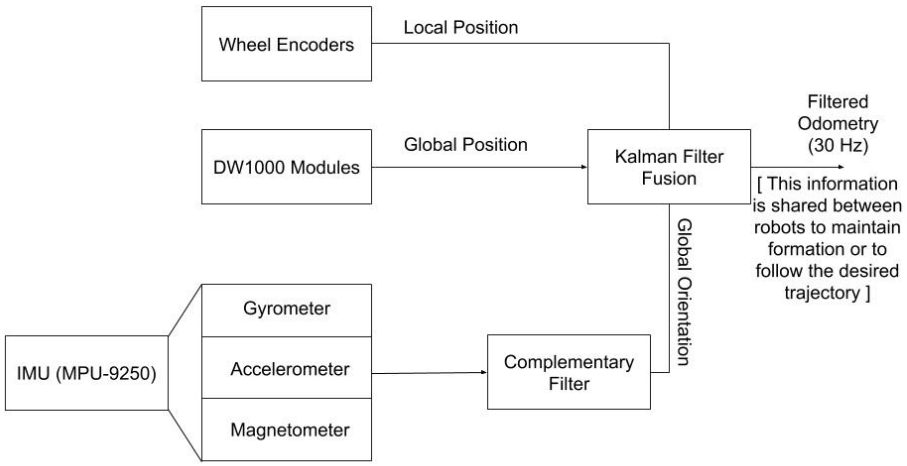


Figure 4: Block diagram of different localization sources to generate a global pose of the robot

$robot\_localization$ package, follow the steps below:

- Install the $Robot\ \ Localization$ package.

- Go to $RobotLocalization$ in aarg github directory.
- Open a terminal and type ***roscd robot_localization***.
- Copy all the files in $RobotLocalization$ to the directory you opened in the terminal.
- Run the launch file by **roslaunch robot_localization ekf_template.launch**

The file will generate *f_odom* topic of message type $Odometry$.

- **Input File:** *ekf_template.yaml* (Check the file name for respective robot id)
- **Publisher:** *f_odom*

Currently there are only 3 robots added in the *ekf_template.launch*. You can add any number of robots as nodes in the file by changing the **namespace** and the input **yaml file**.
Note 1: Without running this launch file, you wont be able to get *f_odom* topic of any robot.
Note 2: We use the robot localization package to fuse the wheel odometry data, IMU data and Decawave Odometry to find a global pose of the robot. More sensors can be added and changes can be done in *ekf_template.launch*. The $Tutorial$ explains the robot localization usage in detail.

10 IMU9250 Calibration

To calibrate the IMU placed on each robot,

- Go to $Robot\_Code/Calibrate$
- Run the *read9axis.py* file.
- Rotate the robot in every possible direction at a slow speed until the counter on the screen is done (Typically set to *120 seconds*).
- Write the generated soft and hard calibrated values in the $imu9250.py$ (Line 14-15)

Hurray, Your IMU is now calibrated.

11 Experiments

These are few experiments to run on the robot. This explains the files we need to run to execute the given task.

11.1 Move a Robot

AIM: We need to move the robot at a particular velocity as given by the user.
The steps are as follows:

- Power ON the robot.
- Login using the credentials (as mentioned above)
- Set the ROS_MASTER and ROS_IP.
- Go to **catkin_ws** directory
- Source the package using ***source devel/setup.bash***
- Launch the file ompl.launch (***roslaunch pibot ompl.launch***)
- Publish the **cmd_vel** for the robot using
  rostopic pub -r 10 /cmd_vel geometry_msgs/Twist
  "linear:
  x: 0.1
  y: 0.0
  z: 0.0
  angular:
  x: 0.0
  y: 0.0
  z: 0.5"
  The linear velocity of the robot should be provided in the **linear (x)**, while the angular in the **angular (z)**. Here **-r** is the rate at which the command velocities should be published. Here, the value of rate is chosen as 10 for illustration purpose. If the rate parameter is not provided in the command then, the value is published on once.

11.2 Manual control of a robot using a keyboard

AIM: We need to move the robot manually using a keyboard.
The steps are as follows:

- Power ON the robot.
- Login using the credentials (as mentioned above)
- Set the ROS_MASTER and ROS_IP (as discussed above).
- Go to **catkin_ws** directory
- Source the package using ***source devel/setup.bash***
- Launch the file ompl.launch (***roslaunch pibot ompl.launch***)
- Run the **pibot_teleop.py** (**rosrun pibot pibot_teleop.py**)
- Now you can use your keyboard to run the robot.
  - **w: Move Forward by 0.05 m/sec**
  - **d: Turn Right by 0.1 rad/sec**
  - **a: Turn Left by 0.1 rad/sec**
  - **s: Move Backward by 0.05 m/sec**
  - **x: Stop the Robot**

11.3 Controlling a Piston of a Robot

AIM: Control the Piston Movement of the robot (Move UP/DOWN)
The steps are as follows:

- Power ON the robot.
- Login using the credentials (as mentioned above)
- Set the ROS_MASTER and ROS_IP (as discussed above).
- Go to **catkin_ws** directory
- Source the package using ***source devel/setup.bash***
- Launch the file ompl.launch (***roslaunch pibot ompl.launch***)
- Run the **broadcast_piston.py** (**rosrun pibot broadcast_piston.py**)
- Make sure to make following changes in the **broadcast_piston.py** file
  - Set the **piston_status** to **MOVE_UP, MOVE_DOWN, STOP** as required.
  - Set the **robot_ids** to a list of all the robot ids, you want to move the piston for.

11.4 Go to Goal

AIM: Move a robot to a goal point
The steps are as follows:

- Power ON the robot.
- Login using the credentials (as mentioned above)
- Set the ROS_MASTER and ROS_IP (as discussed above).
- Go to **catkin_ws** directory
- Source the package using ***source devel/setup.bash***
- Launch the file ompl.launch (***roslaunch pibot ompl.launch***)
- Run the Robot localization package (see section 9)
- Run the **ompl_demo.py** (**python ompl_demo.py**)
- Make sure to make following changes in the **ompl_demo.py** file
  - Set the Goal point

11.5 Move the robots in Formation

AIM: To make the robots move in a rigid formation.
For example: In this case, we consider **Robot ID: 1** as *Leader* and **Robot ID: 2** as *Follower*. This will make our illustration easy.

The steps are as follows:

- Power ON the robots.
- Login using the credentials (as mentioned above)
- Set the ROS_MASTER and ROS_IP (as discussed above). (Make your laptop as master and all the robots as slave)
- Go to **catkin_ws** directory
- Source the package using ***source devel/setup.bash***
- Launch the file ompl.launch (***roslaunch pibot ompl.launch***) for the leader.
- Launch the file pibot.launch (***roslaunch pibot pibot.launch***) for the follower.
  Note: Before running this file, open the file in an editor, go to formation node, and change the **leader id to 1**, **Lij, phi_ij to a value you want the follower to maintain from the follower**. Do not change the other parameters of the file.
- Run the Robot localization package (see section 9)
- Publish the **cmd_vel** only for the leader robot using
  rostopic pub -r 10 /cmd_vel geometry_msgs/Twist
  "linear:
  x: 0.1
  y: 0.0
  z: 0.0
  angular:
  x: 0.0
  y: 0.0
  z: 0.5"
  As soon as the velocities of the leader start publishing, the follower(s) will start following the leader. The formation will automatically stop once the command velocities of the leader stop publishing.

Bibliography

[1] "Fire bird 5 atmega 2560." http://www.nex-robotics.com/products/fire-bird-v-robots/fire-bird-v-atmega2560-robotic-research-platform.html. Accessed: 2019-03-05.

[2] "Turtlebot burger." http://www.robotis.us/turtlebot-3-burger-us/. Accessed: 2019-03-05.

[3] "Turtlebot 2 base kit." https://www.sgbotic.com/index.php?dispatch=products.view&product_id=2408. Accessed: 2019-03-05.

[4] "Pioneer-3dx robot." https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html. Accessed: 2019-03-05.

[5] "Husky robot." https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/. Accessed: 2019-03-05.