INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

International Institute of Information Technology

Agents and Applied Robotics Group

Minion 2.0 Manual

Author:
Rishabh Dev Yadav

Supervisor:
Prof Kamalakar Karlapalem

Instructions for Robot Codes

March 5, 2020

Contents

1 IMU files **2**

2 Complementary Filter **2**

3 Decawave Module Localization & Transformation **3**

4 Camera Localization & Transformation **4**

5 Robot Localization Package **4**

6 ROS Arduino Bridge **5**

7 Quick Overview of packages **5**

7.1 RapsbeeryPiCode 5

7.2 LaptopCode 6

8 Quick Overview of launch file **6**

8.1 RaspbeeryPi/pibot/launch 6

8.2 LaptopCode/pibot/launch 6

9 Decawave Module **6**

9.1 Hardware 6

9.2 Anchor 7

9.3 Tags 7

10 Other Codes **8**

10.1 Leader Follower Code 8

10.2 OMPL Code 8

1 IMU files

1. MPU9250.py
Class for MPU9250 I2C protocol.

2. fusion.py
Class provides sensor fusion allowing heading, pitch and roll to be extracted. This uses the Madgwick algorithm. The update method must be called periodically.

3. read9axis.py
It imports fusion.py and MPU9250.py class mentioned above.
Run this file and swing the magnetometer in shape of Infinity gently for 120 seconds.
It will give MAG_HARD_BIAS and MAG_SOFT_BIAS vectors, copy it and paste the vector in imu9250.py file

4. imu9250.py
It imports MPU9250 class.
This file read the raw data from IMU sensor.
First calibrate and modify MAG_HARD_BIAS and MAG_SOFT_BIAS. Eg:-
MAG_HARD_BIAS = (113.99, -40.54, -16.35)
MAG_SOFT_BIAS = (0.96, 0.98, 1.06)

Publisher:
1. rospy.Publisher('imu/data_raw', Imu, queue_size=10)
Contain the ax, ay, az from accelerometer and gx, gy, gz from gyroscope

2. rospy.Publisher('imu/mag', MagneticField, queue_size=10)
Contains the mx, my, mz from magnetometer

2 Complementary Filter

First install the imu_tools package to your src folder.
See http://wiki.ros.org/imu_complementary_filter.
It fuses the accelerometer, gyroscope, magnetometer raw data to give roll, pitch, yaw.
This node is executed in launch file, there is no separate python file.

Suscriber:
1. imu/data_raw, Imu
Message containing raw IMU data, angular velocities and linear accelerations.

2. imu/mag, MagneticField
[optional] Magnetic field vector.

Publisher:
1. imu/data, Imu
The fused Imu message, containing the orientation.

3 Decawave Module Localization & Transformation

1. get_distances.py
This will read distances vector (Anchor1, Anchor2, ...)
Publisher:
1. rospy.Publisher('distances', Float32MultiArray, queue_size=10)

2. trilateration_optimise.py
Subscriber:
1. rospy.Subscriber('/distances', Float32MultiArray, cbDistances)
Perform optimization and convert it into x,y state.

2. rospy.Subscriber('/imu/data', Imu, cbHeading)
Get the IMU fused information from Complimentary filter node and give theta state

Publisher:
1. rospy.Publisher('heading_deg', Float64, queue_size=10)
Give heading into degree. (180 to -180)
2. rospy.Publisher('imu_fuse', Imu, queue_size=10)
Convert the heading into quaternion.
3. rospy.Publisher('d_odom', Odometry, queue_size=10)
Contains Position x,y, z=0 from Decawave and quaternion orientation w, x, y, z from IMU fused/filtered data.

Note:
It is recommended to run trilateration.py and complimentary filter node on PC instead of Raspberry Pi. optimization operation. So for each robot one trilateration.py will be executed from script folder of PC.

Total frames:
1. world
2. {}_initial_pose :- Initial starting Pose of robot
3. {}_base_link_wheel :- Pose of robot by wheel encoder value
4. {}_kalman :- Kalman filter Pose (fused data of imu, decawave, encoder)
5. {}_Deca : - Decawave module place on robot x,y state

1. tf_fodom_world.py
"world" and "{}_kalman" transformation
Subscribed Topic:- f_odom, Odometry

2. tf_stp_odom.py
"world" and "{}_initial_pose" transformation
Subscribed Topic:- d_odom, Odometry

3. tf_world_deca.py
"world" and "{}_Deca" transformation
Subscribed Topic:- d_odom, Odometry

4. base_controller.py (inside ros_arduino_python folder)
"{}_initial_pose" and "{}_base_link_wheel" transformation
Subscribed Topic:- odom_wheel, Odometry

4 Camera Localization & Transformation

First run the Motion Capture System and Cos Phi.
Publisher: rospy.Publisher('/Robot_poses', CamPoseArray, queue_size = 10)

Total frames:
1. world
2. {}_initial_pose :- Initial starting Pose of robot
3. {}_base_link_wheel :- Pose of robot by wheel encoder value
4. {}_pose :- Camera Pose

1. sus_pub.py
change "camID = X", the id number placed on robot. It will fetch particular camID state and save it to " ns/f_odom,Odometry"
Suscriber:
1. rospy.Subscriber('/Robot_poses', CamPoseArray, callback)

Publisher:
2. rospy.Publisher(/f_odom, Odometry, queue_size = 10)

2. s_tf.py
"world" and "{}_initial_pose" transformation
Subscribed topic: f_odom, Odometry

3. d_tf.py
"world" and "{}_pose" transformation
Subscribed topic: - f_odom, Odometry

5 Robot Localization Package

This is used for data fusion of Encoder, IMU, Decawave using Kalman Filter.
For installation see http://docs.ros.org/melodic/api/robot_localization/html/index.html
This node will be executed by launch file and all the parameters will written in robot_localization/params/ekf_template.yaml.

Suscriber:
1. odom_wheel, Odometry()
2. d_odom, Odometry()
3. imu_fuse, Imu()

Publisher:

1. f_odom, Odometry()

Note:
This node must be executed on PC instead of Raspberry Pi. So for each robot one ekf_template.yaml file will be there.

6 ROS Arduino Bridge

Install ROSArduinoBridge from https://github.com/hbrobotics/ros_arduino_bridge
Default code is for 2 wheeled robot, but here for the 4 wheeled robot, required modification has been done in ".ino" files (See ArduinoMegaCode folder) and ros_arduino_python (See RaspberryPiCode/ros_Arduino_bridge).
Upload Arduino Mega Code ".ino" files to Arduino mega board.

In ros_arduino_python /config/my_arduino_params.yaml file. Set the parameters.
BE CAREFUL !!! Port: USB0 or ACM0 (hit and trial for different arduino boards)

In ros_arduino_python three file are there
1. arduino_drive.py
2. arduino_sensors.py
3. base_controller.py
This file calculate robot kinematics and uses encoder values for position, orientation and velocity calculation.
Publisher:
1. rospy.Publisher('wheel_odom', Odometry, queue_size=5)
Give the state and velocity of robot using only encoder counts.

Subscriber:
1. rospy.Subscriber("cmd_vel", Twist, self.cmdVelCallback)
Take (vx, vy, vz wx, wy,wz) linear and angular velocity to robot.

7 Quick Overview of packages

7.1 RapsbeeryPiCode

1. custom_msgs : -used for camera localization
2. ros_arduino_bridge: - for flow of data between Pi and Arduino. Solve kinematic, encoder counting, subscribe "cmd_vel, Twist()" for movement of robot
3. pibot: - contain our custom launch file and scripts. It read the data from IMU, Decawave Module, Tran formations, Camera pose value.

7.2 LaptopCode

1.imu_tools: -used for quaternion based sensor fusion and complementary filter. IMU raw data read by imu9250.py file of RaspberryPi/pibot package and here publish the filtered data.
2.robot_localization: - Kalman filter for sensor fusion
3.teleop_twist_keyboard: - manual movement of robot using keyboard.
4. pibot: - calculate the decawave trilateration optimization on PC instead on Pi itself.

8 Quick Overview of launch file

8.1 RaspbeeryPi/pibot/launch

1. run ros arduino bridge
If using IMU
2. run imu9250.py to read imu raw data from MPU9250
3. run static transformation between imu frame(original earth frame)
If using the Decawave then
4. run get_distances.py to read distances from each anchor to robot.
5. run transformation files; tf_stp_odom.py, tf_world_deca.py, tf_fodom_world.py
If using Camera localization
6. run sus_pub.py, s_tf.py, d_tf.py

8.2 LaptopCode/pibot/launch

If using IMU
1. run imu_complementary_filter node
If using Decawave
2. run trilateration_optimise.py
For sensor IMU, Encode, Decawave data fusion
3. run robot_localization node for kalman filter
If manual movement of robot
4. run teleop_twist_keyboard.py

9 Decawave Module

9.1 Hardware

Decawave modules are mounted on Arduino Pro-Mini. First install the arduino-dw1000 libraries in IDE and ".ino" will be uploaded in pro-mini using Arduino Uno/Mega.
See https://www.instructables.com/id/How-to-Program-Arduino-Pro-Mini-Using-Arduino-UNO/

9.2 Anchor

Anchors are stationary Modules placed around the arena and its {x,y} coordinates are mentioned in launch file as argument while executing trilateration_optimise.py

Open decapose -> defaultAlgoAnchor -> defaultAlgoAnchor.ino
For each of anchor, whole .ino code will be remain unchanged, just change two variable values for each of anchor:
1. int anchorId
2. int antennaDelay

For antena 1, anchorId = 0. For antena 2, anchorId = 1 and so on.
antennaDelay value is different for each module. So, see the Module number written backside of it and use the following antenaDelay value.
DECA 1 : 16465
DECA 2 : 16465
DECA 3 : 16458
DECA 4 : 16469
DECA 5 : 16490
DECA 6 : 16454
DECA 7 : 16462
DECA 8 : 16465
DECA 9 : 16458
DECA 10 : 16468
DECA 11 : 16458
DECA 12 : 16446
DECA 13 : 16450
DECA 14 : 16454
DECA 15 : 16452

1 dot : 16445
2 dot : 16419
3 dot : 16440
4 dot : 16463
Blue : 16466

9.3 Tags

Each robot has a module, let's call it as Tag, which will receive a distance array from all the anchors. See the manual of Decawave different "enableMode" are available.

Open decapose -> defaultAlgoBot -> defaultAlgoBot.ino
For each of tag, whole .ino code will be remain unchanged, just change two variable values for each of tag:
1. int noOfAnchors = 3;
2. int antennaDelay = 16450;
3. int tagId = 24;

"noOfAnchors" is total number of anchors used. Ideally use 3 anchor for localization, because our trilateration optimization code is for 3 Anchor.
"antennaDelay" different for each Modules, see the table.
If robot 1st, tagId = 21. For robot 2nd, tagId = 22 and so on. Depending upon how many robots has been used, tagID will always start from 21.

Note:
Increasing the number of robot, will decrease the accuracy. Usually use 3 anchor and 3 robot for good accuracy. But if number of robots will be more than 3 then different channels or cluster has been made. So, each of 3 anchor and 3 tag has has same channel; other 3 anchor and 3 tag have different channel.

To add:
OMPL
Motion Capture System

10 Other Codes

10.1 Leader Follower Code

Python File: RaspberryPiCode/src/scripts/lf_bio.py
It will subscribe the leader's robot Position, Orientation and Velocity. Also, it subscribe it's own current position and orientation and publish the required velocity for the follower robot.
Subscriber:
1. rospy.Subscriber('/'+self.LeaderName+'/cmd_vel', Twist, self.twistCallback)
2. rospy.Subscriber('/'+self.LeaderName+'/f_odom' , Odometry, self.leaderOdomCallback)
3. rospy.Subscriber('d_odom', Odometry, self.currOdomCallback)

Launch File: RaspberryPiCode/src/launch/minion2.launch
Here we will pass some parameters which are as follow:
1. "leader" -> node name of leader robot
2. "L_ijd" -> desired distance between leader and follower
3. "phi_ijd" -> desired angle (degrees) between leader-follower
3. "k1","k2","k3","A","B","D" -> parameters, tune it.

10.2 OMPL Code

This is known as Open Motion Planning Library provide consist path planning algorithm like A*, D* and obstacle avoidance. It's installation might take 10-15 hours. Be aware of python version in OMPL and python version of ROS, both should be of same version.
See : https://ompl.kavrakilab.org/index.html

Installation : https://ompl.kavrakilab.org/installation.html
To run this file, ompl should be installed on the PC (install ompl app python bindings version). The file uses the Pose.txt file.
Python files:

1. cubic_spline_planner.py ; Class used for trajectory generation and smoothing
2. go_to_goal.py ; important points to be noted are:

1. Variable "pose_path" to import the .text file
2. module "pure_pursuit_control(*)" uses the controller.
3. "cubic_spline_planner.calc_spline_course" mainly used for subdivision of way points and smoothing.
4. Subscriber : take it's current position and orientation
rospy.Subscriber('/f_odom', Odometry, update_pose)
5. Publisher : give the desired velocity for tracking
rospy.Publisher('/cmd_vel', Twist, queue_size=10)
6. variable "def_points" is defined for obstacle size.
7. .txt file; Since in-multi-agent one robot is obstacle for other robot so, it contain info of obstacle(other robot) position with robot ID,robot initial state and final state and it's ID $\{x,y,\theta\}$