SIT310 – Task 7
Advanced Sensors

Overview

This task focuses on utilising more advanced sensors. We will be learning to use a, magnetometer. We will use this data in ROS to improve visualisation and decision making.

Task requirements

a. Ensure you have already completed tasks 1-6 and assessments 1 - 3.
b. To complete this task, you will need to use ROS on the raspberry Pi as we will be using your physical robot
c. If you didn't attend the lecture this week, then either watch the lecture online and/or review the PowerPoint slides on the unit site.
d. Read the task instructions
e. Complete the activities in the task guide
f. Review your progress with your lab tutor or cloud tutor.

Task guide

So far, we have created been using the commands we send to the robot to move in order to infer the robot's position. This is generally reasonable, but needs to be calibrated to the floor types, and will drift over time. This task will use the Zumo magnetometer to improve the accuracy of our positioning. This data can be fed into transformations and rviz to improve visualisations and decision making.

1. Using the Magnetometer as a 3D Compass

    1.1. So far, we have been inferring the robot's position based on the commands we send to the robot to move. The Zumo also contains a ST Microelectronics LSM303D chip that contains a 3D Compass (Magnetometer). We can use this to more accurately infer the position of the robot.

    1.2. Details of the LSM303D chip are available on ST Microelectronics website at: https://www.st.com/content/st_com/en/products/mems-and-sensors/e-compasses/lsm303dlm.html

    1.3. The datasheet here https://www.pololu.com/file/0J703/LSM303D.pdf contains a ton of interesting information. Pololu also have information about a breakout board they produce with this chip: https://www.pololu.com/product/2127 Have a look at these until you are comfortable with the idea of the module.

    1.4. The magnetometer will give readings for the strength of the magnetic field in the X, Y and Z axis. Knowing the earth has a magnetic north, this means we can calculate the orientation of the robot.

    1.5. Firstly, we need to calibrate the Zumo's magnetometer as they are sensitive. To calibrate it, we need to get the minimum and maximum values of the magnetometer in the x, y and z axis. Upload the following code to your Arduino.

```
#include <Wire.h>
#include <Zumo32U4.h>

LSM303 compass;

char report[120];

void setup()
{
  Wire.begin();

  if (!compass.init())
  {
    // Failed to detect the compass.
    ledRed(1);
    while(1)
    {
      Serial.println(F("Failed to detect the compass."));
      delay(100);
    }
  }
```

```
  compass.enableDefault();
}

int minx=32767, miny=32767, minz=32767, maxx=-32767, maxy=-32767, maxz=-
32767;

void loop()
{
  compass.read();
  if(compass.m.x<minx) minx=compass.m.x;
  if(compass.m.y<miny) miny=compass.m.y;
  if(compass.m.z<minz) minz=compass.m.z;
  if(compass.m.x>maxx) maxx=compass.m.x;
  if(compass.m.y>maxy) maxy=compass.m.y;
  if(compass.m.z>maxz) maxz=compass.m.z;

  snprintf_P(report, sizeof(report),
  PSTR("Current: %6d %6d %6d    MAX: %6d %6d %6d"),
  compass.m.x, compass.m.y, compass.m.z,
  compass.m.x, compass.m.y, compass.m.z);
  Serial.println(report);

  snprintf_P(report, sizeof(report),
  PSTR("MIN: %6d %6d %6d    MAX: %6d %6d %6d"),
  minx, miny, minz,
  maxx, maxy, maxz);
  Serial.println(report);

  Serial.println();
  delay(100);
}
```

1.6. Open the serial monitor. You will see two lines being repeated repeatedly. The first is the current readings of the magnetometer, the second is the current minimum and maximum values. "Be sure to rotate the sensor slowly about its center so that the accelerometer readings will represent just acceleration due to gravity and not linear acceleration of the sensor due to movement. After a while, the sketch output will stabilize." (https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/calibration) When the MIN and MAX values won't change anymore. Stop and copy the MIN and MAX values.

1.7. We can now get the Zumo to report a compass bearing. The following Ardunino code performs this calculation and outputs the compass bearing. Change the six values for the calibration and upload it to your Arduino. Upload the code to the Zumo. You should see the calculated compass reading in the serial monitor.

```
#include <Wire.h>
#include <Zumo32U4.h>
```

```
LSM303 compass;

void setup()
{
  Wire.begin();

  if (!compass.init())
  {
    // Failed to detect the compass.
    ledRed(1);
    while(1)
    {
      Serial.println(F("Failed to detect the compass."));
      delay(100);
    }
  }

  //replace your calibration values
  compass.enableDefault();
  compass.m_min.x=0;
  compass.m_min.y=0;
  compass.m_min.z=0;
  compass.m_max.x=0;
  compass.m_max.y=0;
  compass.m_max.z=0;
}

void loop()
{
  compass.read();
  Serial.println(compass.heading());
  delay(100);
}
```

1.8. Download a digital compass app for your mobile phone. Test the calibration. How is it?

1.9. Have a look at how it is calculated in ~/Arduino/libraries/Zumo32U4/LSM303.h". Pretty neat.

2. Improving our robot orientation in ROS.

2.1. Now that we can reliably have consistent and accurate robot orientation, we should add it to our ROS application.

2.2. As we are now doing a lot in our Arduino program, we need more RAM on the Arduino. We can decrease the buffers available to rosserial. Edit the file ~/Arduino/libraries/ros_lib/ros.h and replace the typedef in the else clause with the typedef from the first clause. E.g. this one:

typedef NodeHandle_<ArduinoHardware, 6, 6, 150, 150> NodeHandle;

2.3. Now, we need to update our Arduino program. The latest version is from 2.8 in Lab 6. The following updates that code to add support for headings from the magnetometer.

```
#define USE_USBCON
#include <Wire.h>
#include <Zumo32U4.h>
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/Int8.h>
#include <std_msgs/Int16.h>

LSM303 compass;

ros::NodeHandle nh;

Zumo32U4ProximitySensors proxSensors;
bool proxLeftActive;
bool proxFrontActive;
bool proxRightActive;
std_msgs::Int16 heading;
std_msgs::Int8 prox_msg;
ros::Publisher pub_heading("/zumo/heading", &heading);
ros::Publisher pub_left("/zumo/prox_left", &prox_msg);
ros::Publisher pub_frontleft("/zumo/prox_frontleft", &prox_msg);
ros::Publisher pub_frontright("/zumo/prox_frontright", &prox_msg);
ros::Publisher pub_right("/zumo/prox_right", &prox_msg);

void ros_handler( const geometry_msgs::Twist& cmd_msg) {
float x = cmd_msg.linear.x;
float y = cmd_msg.linear.y;
int speed = 100;

if(x == 1.0) forward(speed);
if(x == -1.0) backward(speed);
if(y == 1.0) left(speed);
if(y == -1.0) right(speed);

stop();
```

```
}

ros::Subscriber<geometry_msgs::Twist> sub("/zumo/cmd_vel", ros_handler);
Zumo32U4Motors motors;


void setup()
{
  //setup the compass.
  Wire.begin();
  if (!compass.init())
  {
    // Failed to detect the compass.
    ledRed(1);
    while(1)
    {
      Serial.println(F("Failed to detect the compass."));
      delay(100);
    }
  }


  //replace with your calibration values
  compass.enableDefault();
  compass.m_min.x=-9690;
  compass.m_min.y=-394;
  compass.m_min.z=-3381;
  compass.m_max.x=-2400;
  compass.m_max.y=4435;
  compass.m_max.z=5697;


  nh.initNode();
  nh.subscribe(sub);
  nh.advertise(pub_heading);
  nh.advertise(pub_left);
  nh.advertise(pub_frontleft);
  nh.advertise(pub_frontright);
  nh.advertise(pub_right);

  proxSensors.initThreeSensors();
  uint16_t defaultBrightnessLevels[] = {1,2,3,4,5,6,7,8,9,10};
  proxSensors.setBrightnessLevels(defaultBrightnessLevels, 10);
}


void publishSensorData()
{
  prox_msg.data = proxSensors.countsLeftWithLeftLeds();
  pub_left.publish( &prox_msg);
  prox_msg.data = proxSensors.countsFrontWithLeftLeds();
  pub_frontleft.publish( &prox_msg);
  prox_msg.data = proxSensors.countsFrontWithRightLeds();
```

```
  pub_frontright.publish( &prox_msg);
  prox_msg.data = proxSensors.countsRightWithRightLeds();
  pub_right.publish( &prox_msg);

}


void forward(int time)
{
motors.setLeftSpeed(100);
motors.setRightSpeed(100);
delay(time);
}

void backward(int time)
{
motors.setLeftSpeed(-100);
motors.setRightSpeed(-100);
delay(time);
}

void left(int time)
{
motors.setLeftSpeed(-100);
motors.setRightSpeed(100);
delay(time);
}

void right(int time)
{
motors.setLeftSpeed(100);
motors.setRightSpeed(-100);
delay(time);
}

void stop()
{
motors.setLeftSpeed(0);
motors.setRightSpeed(0);
}

//control timing without pausing program.
    static uint16_t lastSampleTime = 0;
void loop()
{
  ledRed(1);
  ledGreen(0);
  if ((uint16_t)(millis() - lastSampleTime) >= 500)
    {
    lastSampleTime = millis();
    // Send IR pulses and read the proximity sensors.
    proxSensors.read();
    publishSensorData();
```

```
    compass.read();
    //int reading = compass.heading();
    heading.data = compass.heading();
    pub_heading.publish(&heading);
    ledRed(0);
    ledGreen(1);
    }


nh.spinOnce();
delay(1);
}
```

2.4. To test this, run the following.

```
$ roscore
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
$ rostopic echo /zumo/heading
```

2.5. You should see the heading change if you move the robot around.

2.6. Next, we you should integrate this in our transforms. Create a new project for this week as follows.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg lab7 tf2 tf2_ros rospy
Check everything is fine.
$ cd ~/catkin_ws
$ catkin_make
Open a new terminal.
$ roscd lab7
$ mkdir scripts
```

2.7. To integrate compass readings with the rest of our programs just requires changing zumo_tf_broadcaster.py since they all use the /zumo transform. So, let's create a new one in the lab7/scripts folder as follows:

```
$ cd scripts
```

```
$ gedit zumo_tf_broadcaster_compass.py
```

2.8. Use the following code for zumo_tf_broadcaster_compass.py

```
#!/usr/bin/env python
import rospy

# Because of transformations
import tf_conversions

import tf2_ros
import geometry_msgs.msg
from geometry_msgs.msg import Twist
from std_msgs.msg import Int16


#for calculating position
import math
from math import sin, cos, pi

x=0
y=0
th=0.0

def handle_zumo_heading(heading_msg):
    global th
    print(heading_msg.data)#in Degrees
    th = heading_msg.data * math.pi / 180.0
    print(th)


def handle_zumo_pose(vel_msg):
    global x
    global y
    global th
    br = tf2_ros.TransformBroadcaster()
    t = geometry_msgs.msg.TransformStamped()

    print("x: "+str(x))
    print("y: "+str(y))
    print("t: "+str(th))

    #for rotation, 5 degrees is 0.0872665 rads
#    if(vel_msg.linear.y==1.0):  th+= 0.0872665
#    elif(vel_msg.linear.y==-1.0): th-=0.0872665

    if(vel_msg.linear.x==1.0): #this is a movement
        vx = 0.1 #velocity of x, how much we move
        # compute odometry
        delta_x = vx * cos(th)
        delta_y = vx * sin(th)
        delta_th = 0  #no change in angle
```

```
        #add the changes to the values
        x += delta_x
        y += delta_y
        th += delta_th

    elif(vel_msg.linear.x==-1.0): #this is a movement
        vx = 0.1 #velocity of x, how much we move
        # compute odometry
        delta_x = vx * cos(th)
        delta_y = vx * sin(th)
        delta_th = 0  #no change in angle

        #add the changes to the values
        x -= delta_x
        y -= delta_y
        th -= delta_th


    #    y+=vel_msg.linear.y
    t.header.stamp = rospy.Time.now()
    t.header.frame_id = "world"
    t.child_frame_id = "zumo"

    #set x,y,z
    t.transform.translation.x = x
    t.transform.translation.y = y
    t.transform.translation.z = 0.0

    #set rotation
    q = tf_conversions.transformations.quaternion_from_euler(0, 0, th)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    br.sendTransform(t)

if __name__ == '__main__':
    rospy.init_node('zumo_tf_broadcaster')
    rospy.Subscriber('/zumo/cmd_vel',
                     Twist,
                     handle_zumo_pose)
    rospy.Subscriber('/zumo/heading',
                     Int16,
                     handle_zumo_heading)

    rospy.spin()
```

2.9. Test is as follows.

```
$ chmod +x zumo_tf_broadcaster.py
$ roscore
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
$ rosrun lab7 zumo_tf_broadcaster_compass.py
$ rosrun lab5 zumo_controller.py
$ rosrun lab6 zumo_tf_sensor.py
$ roscd lab5
$ rviz -d ./zumo.rviz
```

2.10. Move the robot a little. You should see that the robot will be facing the correct way. As you move and rotate the robot around you will see that the robot's location jumps in rviz. It is doing this as the magnetometer is also detecting linear acceleration as well as the force placed on the robot by gravity. If you move the robot very slowly you will see it working smoothly. You can improve this by either i) moving the robot very slowly. ii) averaging the values coming from the robot so you don't get big jumps and errors, and iii) combining the compass with the known movements – e.g. don't accept it when the compass says it has rotated left 5 degrees, when the robot actually turned right. Spend the remaining lab time this week experimenting with these ideas.

3. Using the Gyroscope

3.1. So far, we have been inferring the robot's orientation based on the commands we send to the robot to move. The Zumo robot contains a ST Microelectronics L3GD20H chip that contains a 3-axis Gyroscope. This can give us accurate readings of orientation.

3.2. Details of the L3GD20H chip are available on ST Microelectronics website at: https://www.st.com/en/mems-and-sensors/l3gd20h.html The datasheet contains a ton of interesting information. Pololu also have information about a breakout board they produce with this chip: https://www.pololu.com/product/2129 Have a look at these until you are comfortable with the idea of the gyroscope.

3.3. Broadly, the L3GD20H is a "three-axis gyroscope, which measures the angular rates of rotation about the roll (X), pitch (Y), and yaw (Z) axes". This means we should be able to determine the robot orientation accurately based on the starting position of the robot. As the gyroscope detects movement not location, this is difficult and would make an interesting project task for you to think about.

4. Additional resources

4.1. Learn about linux

- https://www.computerworld.com/article/2598082/linux/linux-linux-command-line-cheat-sheet.html
-

4.2. Learn about ROS.

- http://wiki.ros.org
- A Gentle Introduction to ROS: https://www.cse.sc.edu/~jokane/agitr/
- http://wiki.ros.org/kinetic/Installation/Ubuntu
- http://wiki.ros.org/ROS/Tutorials
- https://www.pololu.com/docs/0J63/4
- http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup
- http://wiki.ros.org/rosserial_arduino/Tutorials/Hello%20World