Teleoperation of A Robotic Arm Using IMU

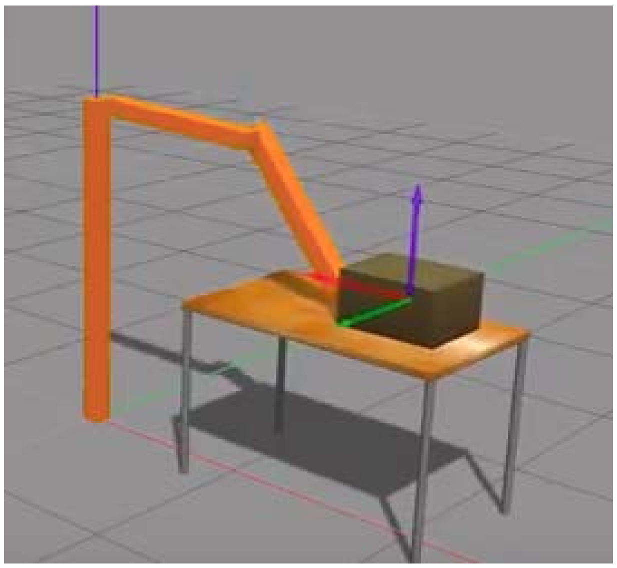
Individual Hands-On Project Description
Student Name

Introduction:

Human safety has always been an issue in hazardous environmental working in any field. To overcome these problems of human safety robot technology can be proved to be a boon. Teleoperation can be a very good way to operate robots from a distance and at the same time keeping the human operator in a safe distant environment. Hence, the most basic way to do this is using an IMU Inertial Measurement Unit, which gives us 9 degrees of freedom. So the flexibility of this particular sensor enables us to operate a robot easily.

Problem Statement:

This task involves a robot in simulation, which is a 4 degree of freedom arm that is operated by the IMU sensor using an arduino board as our controller. This robot is known as the rrbot. We use the gazebo simulator for our simulation and also the Robot Operating System (ROS) for the communication between the IMU sensor and the rrbot. The robot uses the IMU signals to move its arm and push an object of the table.



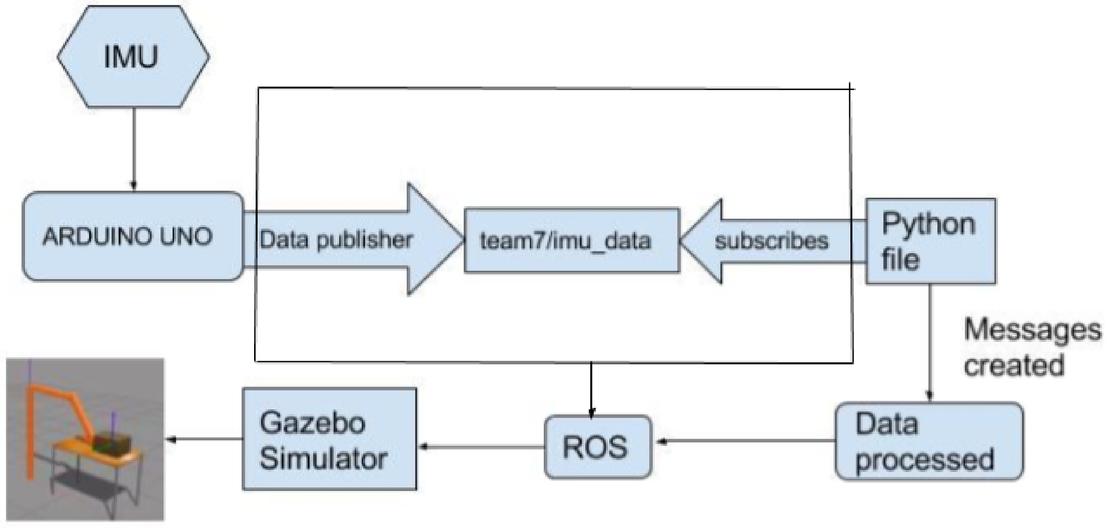
Hardware:

The following hardware components have been used for this project:

1.) Arduino UNO R3
2.) IMU sensor (Accelerometer, Gyroscope, Compass)
3.) Push Button
4.) 10 k Ω resistor
5.) Connecting wires

Software:

We used Gazebo Robot simulator for recording te robot movements when readings are given by the IMU sensor.



High Level Software Architecture of the Program

Program Setup:

At first, the arduino collects the sensor data from IMU sensor
Now, the arduino sketch(c++ code) running on arduino board publish the sensor data in form of a geometry_msg (imu_msg) to the ROS Topic (/team7/imu_data)
The python program (pub_to_arm.py) subscribes to the topic "/team7/imu_data", processes the data and create a message to publish to the ROS topic /rrbot/joint1_position_controller/command
Over the ROS controller package, this message simulates the robot arm movement in Gazebo

Summary of the Project Requirements / Accomplishments

1. Summary of Accomplishments:

1.1. Did you complete all of the basic requirements? Yes.
The electronic components that we used are:
Button
IMU
Arduino Uno R3

1.1.1. Download latest Arduino IDE from arduino.cc.
Yes. We downloaded the IDE and successfully installed t on Ubuntu OS
1.1.2. Learn Arduino C/C++ and its IDE.
1.1.3. Learn about interfacing with a variety of sensors. (Analog, PWM, Interrupts, SPI, I2C).
1.1.4. Write a simple sketch to get an Arduino to interface with a sensor.
1.1.5. Show that Arduino can properly handle interrupts.
1.1.6. Get Arduino to communicate with a laptop using the Arduino serial class.
1.1.7. Get Arduino to send ROS messages of sensor outputs using Arduino ros-serial library.
1.1.8. Learn Ubuntu (Linux) and its tools.
1.1.9. Learn the basics of Python.
1.1.10. Learn the basics of the Robot Operating System (ROS).
1.1.11. Install Ubuntu (14.04 works) and ROS (Indigo) on Laptop
1.1.12. Have ROS store sensor data for later analysis. Probably use ROS "bags".
1.1.13. Analyze the data using Python.\
1.1.14. Understand the characteristics of sensors. Don't just connect them up. Document them:
IMU:

The inertial measurement unit (GY-85) which we have is a 9 DOF module and works by detecting linear acceleration using one or more accelerometers, rotational rate using one or more gyroscopes and includes a magnetometer which is commonly used as a heading reference. This unit contains one accelerometer, gyro, and magnetometer per axis for each of the three vehicle axes: pitch, roll and yaw.

Arduino UNO R3:

The Arduino Uno R3 is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

1.1.14.1. Their interface characteristics and code.

The IMU connects via I2C protocol and the Wire.h library was used for Arduino.

The IMU unit's supply voltage range is from 2.0 V to 3.6 V
ADXL345 - 0x53 — Three axis acceleration
ITG3205 - 0x69 — Three axis gyroscope
HMC5883L - 0x1E — Three axis magnetic field

Arduino UNO R3:
The operating voltage and temperature ranges from 1.8-5.5V -40 C to 85 C respectively.
Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

AREF. Reference voltage for the analog inputs. Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

1.1.14.2. Their error characteristics and statistics.

Gyroscopic and accelerometer sensors behavior is represented via a model based on following errors:
Offset error: this error can be split between stability performance (drift while the sensor remains in invariant conditions), and repeatability (error between two measurements in similar conditions separated by varied conditions in between)

- scale factor error: errors on first order sensitivity due to non repeatabilities and non linearities
- misalignment error: due to imperfect mechanical mounting
- cross axis sensitivity: parasitic measurement induced by solicitation along an axis orthogonal to sensor axis
- noise: dependent on desired dynamic performance
- environment sensitivity: mainly sensitivity to thermal gradients and accelerations

	Mean	Standard Dev
Linear X	16878.02	16.14
Linear Y	413.2292	16.33343

Linear Z	413.2292	19.47601
Angular X	-160.781	9.57558
Angular Y	-54.7396	7.149075
Angular Z	17.21875	9.135785

1.1.15. Keep notes as you progress.

1.2. Did you accomplish other objectives?

2. Problems Encountered:

-Arduino constantly loses connection over Rosserial (probably because memory was low)

-The sketch was originally too large for the Arduino's memory so some adjustments had to be made to shrink it

-We had trouble finding a good package for a robot model that we could operate. It needed to have good message types that we could access and run on the computers we had.

3. Solutions to (or work-arounds for) problems encountered:

- We stored the data in a ROS bag to avoid the connection problem. This allowed us to play back a constant stream of data.

-The data types in the sketch and messages used just had to be adjusted to conserve memory space

-We found a few robots that met our criteria and settled on the best one, although we have implementations for all 3.

Future Work

In future we would like to accomplish the following:

1. Smooth Movement of the robotic arm to complete basic tasks like picking and placing using an end effector
2. Implementing kalman Filter for filtering out noise from the data and achieve smoother/precise movements
3. Using multiple Degree of Freedoms of the robotic arm using more than one IMU sensors
4. Simulate the robotic arm movement in more complex environments for example shutting down the power supply in case of a disaster/catastrophes