**Robot Control Interface and Code**

**Documentation v 1.0**

**0. Installing and General Information**

This program is written in C++ using the Qt IDE and framework. The software was tested on Ubuntu 14.04 LTS. Newer versions of Ubuntu should work but they have not been tested.

**0.1 Required Software**

Qt: [http://doc.qt.io/qt-5/linux.html#downloading-and-installing-qt](http://doc.qt.io/qt-5/linux.html" \l "downloading-and-installing-qt)

I found that the installer made things easier.

Other Requirements:

sudo apt-get install build-essential libgl1-mesa-dev

Import project in Qt Creator:

Import Project → navigate to and open “<project\_name>.pro” file within <project> folder

Gyroscope - Phidget Sensor <http://www.phidgets.com/docs/OS_-_Linux>

The steps listed in link are as follows below:

1. Install libusb-1.0 development libraries - libusb-1.0-0-dev.

Note that libusb-1.0 may be already on your system, but the development libraries probably aren't.

sudo apt-get install libusb-1.0-0-dev

2. Unpack and install the Phidget Libraries

To Unpack run the following:

tar -xvzf libphidget\_2.1.8.20151217.tar.gz

From the main unpacked libraries directory, run:

./configure

make

sudo make install

This will compile phidget21.h and place the library into your gcc path

To enable the usage of the phidget sensor without needing sudo (hence also needing to have the robotcontrol running as sudo), there needs to be a modification to the udev rules. When in the libphidget folder run the following command:

sudo cp udev/99-phidgets.rules /etc/udev/rules.d

This will update the usb rules to allow for the phidget to be accessed without needing sudo.

(More Info in the README file in libphidget folder)

Player: <http://playerstage.sourceforge.net/>

You will need to get the laser driver ( hokuyoaist) to read from the laser sensor. This can be done in various ways. The method chosen was to download the latest version of gearbox, this however requires the svn version of player.

Prerequisites: (In-case cmake and cpp are not installed)

sudo apt-get install cmake cpp

Laser ( hokuyoaist) && Flexiport (dependency for hokuyoasit)

DO FOR BOTH HOKUYOASIT && FLEXIPORT

1. Navigate inside folder.

We make a new folder to place the compiled version

mkdir build

cd build/

2. We create the make file

cmake ../

ccmake ../

navigate to BUILD\_DOCUMENTATION

[enter]

to change from ON to OFF

[c]onfigure

[g]enerate

3. Then we build and then install

sudo make

sudo make install

Test Laser ( 'playerv' can also be used to visualize data from blobfinder )

First test player

player

1. Navigate to src/player/config

player roomba.cfg

2. In a seperate terminal navigate to player/examples/libplayerc++

sudo playerv

3. A window will pop up, click on the menu named 'Devices' → 'ranger:0 (hokuyoaist)' → 'Subscribe'

Project Notes: The SVN version of player is using a depreciated type in boost signals. Thus there will be warnings appearing when compiling of the project. The non SVN version of player can be used, but changes will need to be made to the make files for the cmake and boost versions as well as changes to the player source files if a newer version of boost is being used. In addition flexiport and hyoukoaist drivers must be found and compiled.

**1. Overall Design**

The code is separated into two distinct versions, the interface that the user interacts with and the control code that controls the robot. The system architecture is shown below.

Figure 1.1 Overall System Architecture

UDP

Robot Control

User Interface

In the most general view the user interface and robot control communicate over UDP. The communication is implemented using code from Dr. Tang's 599 class. The robot computer and the user interface computer are connected over an ad-hoc network. The network is a hot spot created on the user interface computer.

Figure 1.2 User Interface Architecture

Alerts/Robot Data

Robot Control

Communication

Commands/Requests

Update Send Commands

Map Display

Vehicle Interface

Vehicle Interface

The user interface receives alerts and robot data from the robot. Robot data currently consists of robot position, waypoint positions, goal position, robot mode, and laser sensor data. The alerts consist of only invalid laser data. The commands are currently new goal position, and mode change. The request is only laser data. The robot position, waypoint positions, and goal position are visualized on the map. The robot position, goal position and mode are shown on the robot control interface. The mode is displayed in text when the robot is not selected and when the robot is selected the mode button is depressed. Alerts are displayed in two locations in the vehicle interface as shown in the image.

(Add image).

The robot interface can send mode change commands, new goal position, resolve errors and request for laser data. mode change commands, new goal position, resolve errors can only be sent when the robot is selected, but laser requests can be done regardless of whether or not the robot is selected.

Figure 1.3 Robot Control Architecture

Commands/Requests

Communication

User Interface

Alerts/Robot Data

Receives Command/Request Send Data

RobotController

Localization

Ranger(Laser)

Navigator(Waypoint)

The robot control code receives commands and requests from the user interface and sends alerts and robot data. Robot data currently consists of robot position, waypoint positions, goal position, robot mode, and laser sensor data. The alerts consist of only invalid laser data. The commands are currently new goal position, and mode change. The request is only laser data. When receiving a new goal position, the current path to goal and goal are removed and a new path is calculated, sending the path to the user interface. For mode change, the robot's actions will change. Autonomous will reset the settings to allow for the robot to move automatically to the goal. Semi-autonomous will allow for the robot to receive new goal commands. Teleoperation will stop auto movement to next waypoint and enable the receiving of manual commands. Peer to Peer will stop the robot from moving automatically to the next waypoint if its position is updated. If the robot has a laser error, it will stop localization send the error message and wait for a resolution. Upon receiving the resolution it will resume localization. The laser data is sent piecemeal. As the size of the data is much greater than the size of the message it is separated on the robot's end. The laser data reading and localization are performed in separate threads. A more detailed description and other possible implementations will be provided in Section 2.

Information being sent and received can be expanded upon and the communication wrapper can be modified to support multiple robots. This will be detailed in Section 2.

**2. Class Breakdown and Interaction**

2.1 User Interface

**Action List – List of Actions for semi-autonomous mode**

-Current Actions: Go to Position, New Robot Position, Video Request

Notes: New Robot position is present but not fully tested, video request is a skeleton on both user interface and robot control needing code to be inserted.

**Alert – Data Class for alerts**

-Stores: Robot that the alert is for, whether the alert is a warning or an error, and the description of the alert.

**AlertGUI – View Class for alerts**

-Stores: Alert being displayed as well as all visualizations needed to display and resolve alert

**Cell – Data Class for map**

-Stores: Information about position in the map, its position, state (i.e. obsticle, robot, etc.), and GUI display position for improved memory.

**Communicate – Communication code**

-sends messages using IP and port

-receives char messages and passes to wrapper to convert and send to GUI

**core – Base of program**

- Stores:Statically declared map, code to read map from file, network wrapper, GUI and list of robot interfaces

-Connects callbacks from network wrapper and GUI to send/receive messages

**DescribeXY – GUI to display x and y positions**

-Stores: X and Y position and description

**GoTo2D – GUI to send (x,y) to robot**

-Stores: x and y position being sent

-Checks to see if input is correct

**GUI – Main interface class**

-Stores:all display interfaces

-Passes all user commands and requests “up” to the network wrapper via core class

-Passes all received robot data and alerts to the interface to update display and visualize

-Note: This is where modification is needed to handle multiple robots. The code needs to allow for multiple vehicle interfaces to send and receive robot data and alerts. Connects could be simplified to be simply message with the id to reduce the number of connect functions. A marshaller class could be created to handle the interfaces, but a simple solution would be to have a vector of vehicleInterfaces. Iterate over the vector during the initialization, having connect for each element. (i.e. connect(VI[i], signal(), this, slot())). New added vehicles would just need to have the connect function run when they are added. This works as the connect function is only needed to be run once in order to connect the signals and slots.

**LaserScene – Display laser data**

-Stores: list of data points and lines for range and display of laser data

-Notes: The display does not update baised upon the orientation of the robot. The up position (90 deg)is the front of the robot. This can be modified if the orientaiton of the robot is also passed. There could be updating issues if the robot is turning too quickly so passing the orientation when sending a laser data packet could help lower the effect.

**LaserView – GUI to display laserScene**

-Stores:Laser scene

**ManualControl – Not Used**

-Previously used to capture, but that is done in the vehicleinterface

-Notes: Update to display pressed key direction and that operator is teleoperating

**robot – data class for physical robot**

-Stores: ID of robot, port and ip of where to send messages to

-Notes: Has getters and setters. Used as a passing class to update the GUI with any new information from the robot.

**Scene – Visual representation of the map**

-Stores: Visual representation of the map

-Notes: Loads in map class and alerts view of when user clicks. Has updating values for new robot positions as well as for reloading values from map and updating the map and changing the display.

**View – Displays visual representation of map**

-Stores: scene

-Notes: Handles zoom and conversion from clicking on map to (x,y) position. The class acts as a wrapper for scene, passing information to it and receiving from it.

**Vehicle Interface – Interface for individual robot**

-Stores: All GUI buttons/displays for individual robot control

-Notes: Has error checking for user input, has two different sizes (expanded and minimal), sends messages to the robot and handles any specific updates from the robot object.

2.2 Robot Control

**Cell – Data Class for map**

-Stores: Information about position in the map, its position, state (i.e. obsticle, robot, etc.), and wavefront value.

**core – Base of program**

- Stores:Statically declared map, code to read map from file, network wrapper, and robot controller class

-Connects callbacks from network wrapper and robot control class

**data – used for localization algorithm**

-Stores: Pointer to ranger proxy as well as the min/max angle as well as the max index.

-Notes: Max index should be found via the proxy but the value being returned is 0.

**localizer – localization algorithm**

-Stores: Past readings to check if there is an error, localization algorithm, dummy localization algorithm, and print statement for collected “object” positions.

-Note: When there is an error, localization will stop running, will resume once bool is changed

**map – Environment**

-Stores: Map of environment

-Notes: Can gather possible robot locations as well as collecting all obstacles that are in the map that match to the laser sensor data.

**navigator – Creates path for robot**

-Stores: Pointer to robot and map

-Notes: Calculates path and smooths the path, and has bool to not send next waypoint

**phigetspatial – phigetsensor wrapper class**

-Stores: phidget sensor object

-Notes: has start and reset functions

**pilot – Class to determine speed and turning rate of robot**

-Stores: Robot pointer and current position

-Note: Has boolean to stop sending movement to next received waypoint

**priorityqueue - implementation for cells**

-Stores: Priority queue for cells

-Note: Used for waypoint

**rangerreader – gets readings from ranger**

-Stores: Pointer to ranger and refresh rate in seconds)

-Notes: Used to send readings to UserInterface

**rdriver – Wrapper to robot to send move commands**

-Stores: robot pointer

**robot – Class for physical robot**

-Stores: player robot classes (client, 2dproxy, rangerproxy), phidget object

-Notes: can reset phidget sensor when localization is paused.

**Robotcontroller – Controlling Class for Robot**

-Stores: All robot sensor accessors, navigation, pilot, localization, ranger reader, player class, and map.

-Notes: Sets up the connections between the different classes to allow the robot to run. Also changes the states of the classes when there is a robot mode change.

2.3 Communication

**network wrapper – wrapper for network class**

-Stores: Network class

-Notes: Abstracts the network layer from the other classes. Thus it can replace the network code with any other code given that the functions are the same. It creates the thread for the communication.

**Network – sending and receiving messages**

Modified given communication files:

<http://www.cpp.edu/~ftang/courses/CS599-DI/hw/communicate.h>

<http://www.cpp.edu/~ftang/courses/CS599-DI/hw/communicate.c>

-stores: port to receive from, and file descriptors for listening and sending.

-Notes: Parses messages to send to and receive from robot. Sending laser data is currently statically set. Need to update for a more dynamic approach. The numbers were chosen to allow for doubles with values from 00.00 to 99.99. This can be modified but the laserBreakdown array would need to be modified or a dynamic approach would need to be created. By using the laserBreakdown array the function gets the values, places the whole number first, delimits, and then places the remainder. As we know the number of elements per message and delimit even if we have no remainder (i.e. 3.00 => 3$00$) we are able to recreate the decimal values when we receive the message.

**Messageids.h – list of message ids**

-Stores: All of the message types and their id's.

Notes: If multiple robots are being used, there needs to be a slight modification to the message structure to add vehicle identification information. Sending to vehicles does not need to be modified, but to keep consistency of message structure as a whole; it should be modified as well. In the core class for the user interface, there needs to be a method to modify how to handle the connect functions. As there is the additional information regarding from which vehicle we are receiving from, we could resolve the connect functions in the core class, calling the appropriate vehicle in the list (or data structure). The functions do not have to be modified as SLOT functions can be called normally.

2.4 Shared Classes

**point2d – class for 2d point**

-Stores: X and Y positions

-Notes: Has getters, equality operators overloaded and the distance formula

**map – stores map of environment**

-Stores: max row, max col, cell size (how large each (x,y) element is) and error val (error for map)

-Notes: Handles import and growth of map as well as the updating of values for the map

**main – starts the program**

-Stores: Core

-Notes: qRegisterMetaType allows for the data to be passed via signals and slots via the data type provided.

**3. Usages**

3.1 Set Up

There needs to be at least 2 computers, one for the user interface and the other for the robot. There is a debug version of the robot control code so a person can test the interface without physical access to a robot (that is a variable called debug in the code, not to be confused with debug run in Qt). More information is detailed in 3.x. For communication there are two options. One would having the computer with the user interface create a hot spot and have all the robots connect, and the other would have all of the devices connect to a router. The former is what is being used currently. The only issue with the latter is making sure that the messages are able to be sent and received across the network. Note that the IP may differ thus requiring a change in the core values of the user interface and robot control, this is done in core.cpp. When creating robotcontroller in robot control and robot in user interface,the ip followed by the port # is used. To get the ip of the computer, if you have the network info on the top bar, click that. On the drop down menu there is a “connection information” option. Selecting that will give you a window which will give you the IP Address of the computer. Alternatively, you could also type 'ifconfig' into terminal.

3.1.1 Setting up the Roomba

The computer connected to the Roomba needs to have the laser, serial connection to the Roomba, and the usb to the phidget connected to it.

The USB connection for the phidget and the USB for the serial connection to the Roomba are connected to the USB Hub. Only the USB Hub and the Laser are directly connected to the laptop.



*IMG 1: Laser on the front of Roomba, hub with USB for phidget (white with white wire) and USB for Roomba serial connection (black with gray wire) Note that the wires are coiled around the phidget sensor to lessen disruption.*



*IMG 2: Laser and USB Hub connected to the laptop.*

 

*IMG 3: Left Side of Roomba with all connections IMG 4: Right Side of Roomba with all connections*

The wires could be cleaned up by using any sort of string, twist tie, etc. This will allow the wires to lie flat and be more compact, making the laptop placement easier.

3.2 Running

To run the User Interface, open the Qt project and run the project.

To run the robot control code, first start player with the following command:

player roomba.cfg

Then from the Qt RobotControl project, run the project. Unless the Booleans that change the robotcontrol code are set to false, the application will throw an error.

The User Interface should be started before the Roomba. If the interface is not then it is possible to not receive the initial robot position. This can be solved by waving an object in-front of the laser sensor to have it send the position again due to it recalculating.

3.x Debug Robot Control Code

The robot control code has a debug mode. This was created for ease of testing the user interface without the need of a physical robot. All major parts of the robot control code are simulated. The only limitations present are from the localization and laser sensor. These values are randomly generated and thus are hard to provide a more “realistic” test. Localization is currently generated within a box with dimensions staticlaly set. The laser sensor also sends random values, but needs to be more heavily modified than the other debug code in order to run in debug mode. All other code that needs the physical robot, i.e. the robot class, has a boolean isPhysicalDebug to specify which code to run, the physical or the debug.

Note that stage has not been tested. It should be possible given the roomba.cfg and a created world file to create a testing environment. This in combination with the ability of the program to read in an image file, that could be used for the stage map, would allow for improved testing.