**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**

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

Other Requirements:

sudo apt-get install build-essential libgl1-mesa-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 ../

2.a If make file was not made

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

Configure Player

sudo ccmake ../

navigate to ENABLE\_DRIVER\_HOKUYAIST

[enter]

to change from OFF to ON

[c]onfigure

[g]enerate

sudo make

sudo make install

to test player

player

1. Navigate to src/player/config

player roomba.cfg

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

( 'playerv' can also be used to test laser and visualize data from blobfinder )

sudo playerv

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

Download Qt's IDE

<http://www.qt.io/download-open-source/>

- IMPORTANT: during “Select Components” screen of Qt Setup

click dropdown arrow next to “Qt” and select (at least) latest Qt version.

Open existing project in Qt Creator:

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

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**

Phase I

- Initialize trial info (IP addresses of machines will change with network change, how far to push box, will this trial introduce an error).

- Determines which machine (Hub, Robot1, Robot2) is running the code and runs corresponding machine's state machine.

Phase II

- HUB Sends start message to Robot1.

- Robot1 locates the box & sends start message to Robot2 before locating its corner.

- Robot1 waits for message to start pushing.

- Robot2 locates the box, locates its corner, then sends message to HUB that it is ready to begin pushing.

- Robot2 waits for message that Robot1 is ready to push.

- HUB forwards that message. Robot1 sends the Ready message and waits a 'heartbeat' before pushing.

- HUB stands by until it receives task completion/ task error message.

Phase III

# HUB #

- If received message from Robot2 that task completed, end trial.

- If received error, choose which mode of sliding autonomy to use

- if Robot2 does not receive help from HUB after 20 seconds, it pushes box on its own

- 'mixed initiative' interrupts this and reposition Robot2 (optional)

- P2P: tells Robot2 to just push straight, human operator will physically be with Robot2 to push with it

- Teleoperate: HUB will teleoperate the 'malfunctioned' Robot1

- Full-Autonomy: HUB tells Robot2 to just push the box on its own

# Robot2 #

- If box pushed for the distance designated in Phase I, send completion message to Hub and end trial.

- If Robot1 experiences an error, it will notify Robot2 and Robot2 will wait for instruction on how to proceed from HUB.

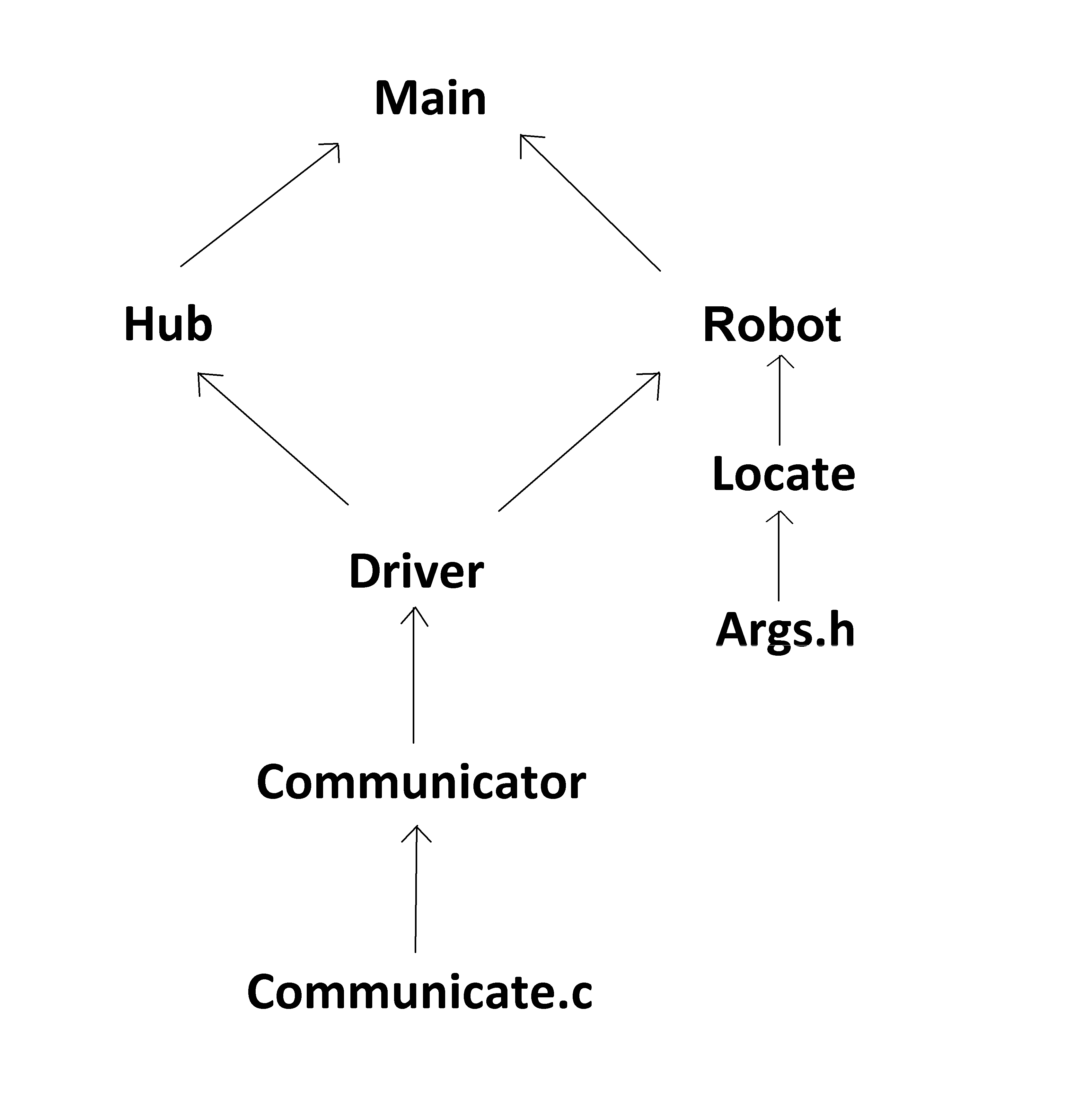
- If after 20 seconds it has not received any instruction, it will enter full-autonomy mode.

- If full autonomy is interrupted, Robot2 will wait for mixed-initiative reposition moves and/or further instruction on how to proceed (p2p, teleoperate, etc)

\* if error introduced, only robot1 needs to know, robot2 will react to robot1's failure

\* Semi-Autonomous -> System Initiative:

-> Mixed Initiative: Hub sends tells Robot which commands to complete (go to left corner of box vs. teleoperation)

**2. Class Breakdown**

**Args.h – Header file from Player/Stage software**

- C code containing libraries for physically moving the robot's hardware (motor, speed, etc.)

**Communicate – Communication code**

- Sends messages using IP and port

- Receives char messages and passes to wrapper

**Communicator - 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>

**-** Wraps the C code from communicate class into c++

- Sends messages

- Parses incoming messages and sets corresponding flags

- small 'system sleeps' so flags are not overwritten before 'Driver' has a chance to grab them.

- Getters for those flags

**Driver – wrapper for Communicator class**

- Wraps communicator into background threads (so robots can perform tasks while communicating)

**Locate – AI logic & Motor functions**

- Contains any method that would have anything to do with moving the robot

- This is the class that uses the 'driver' wrapper

**Main – starts the program**

- Input IP addresses for the 3 computers

- Change the ID associated with the robots/hub (located in the “myID.txt” file inside the project)

- Trial variables (to introduce an error, iterations to push the box for)

**RunasHub – State machine for HUB**

- User input for starting trial & error handling

- Controls for sliding autonomy

**Runasrobot – State machine for Robot1 & Robot2**

- Sliding autonomy for Robot2

- Robot1 forwards messages from HUB to Robot2

- Teleoperation of Robot1

**3. Set UP**

3.1 Communication Network

There needs to be at least 2 computers, one for the user interface and the other for the robot. For communication, the 'Hub' computer needs to create a hot spot and have all the robots connect. Note that the IP will differ based on where you create the hot spot. 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.

**WARNING: everything MUST be hooked up to laptop and turned on BEFORE turning on the laptop. Otherwise 'flexiport' error thrown when trying to run.**

3.2 Running

The Hub computer and two robot computers all run the exact same code – just change the ID associated with the corresponding computer in 'main'.

To run the Hub, open the Qt project and run the project.

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

(the 'roomba.cfg' file should be located on the desktop for easy access to starting player)

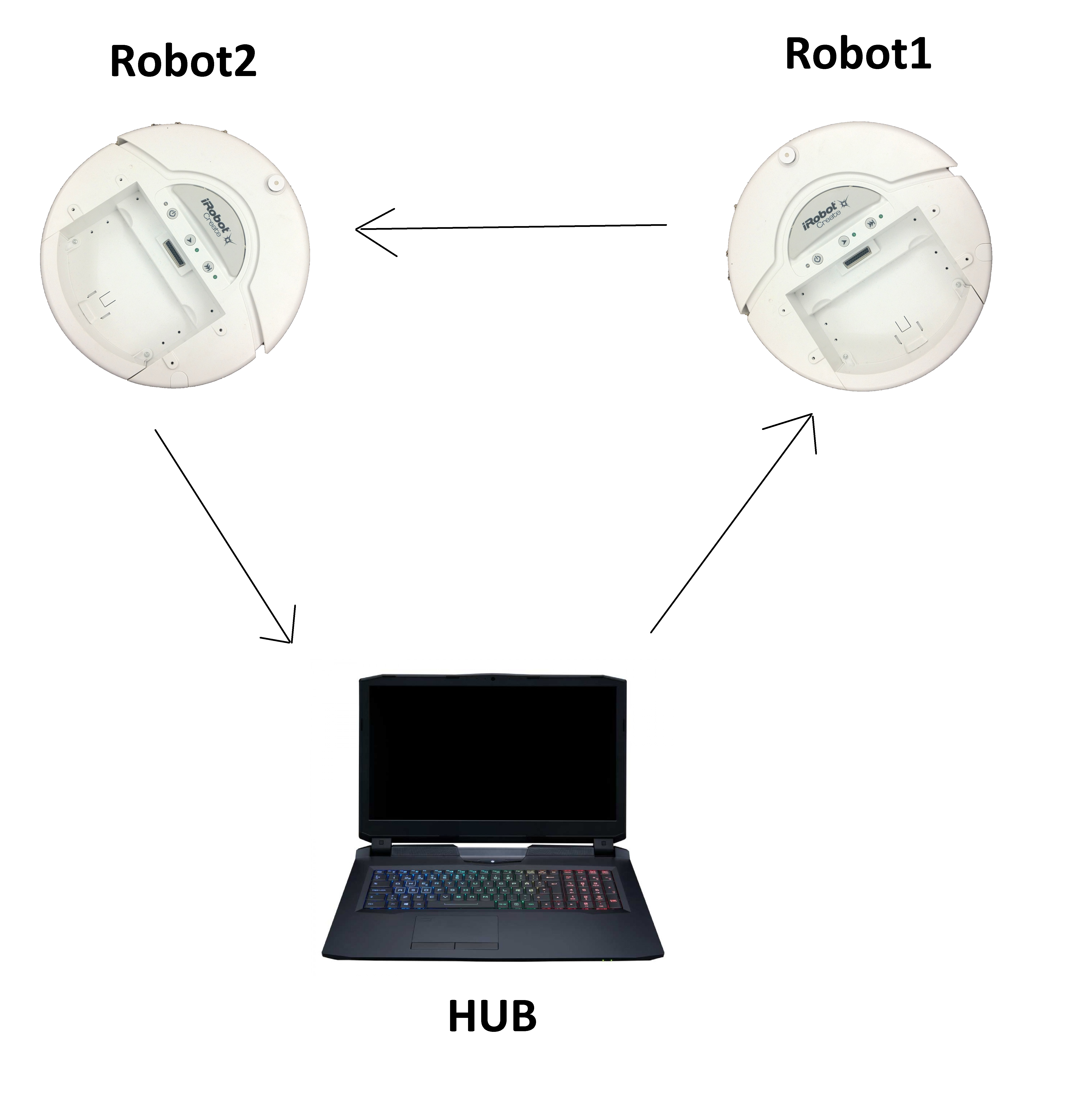
player roomba.cfg

Then from the Qt Communicator project, run the project.

**4. Issues**

4.1 Communication

The way communication works is, when a robot receives a message it then triggers a 'flag'. Each robot is constantly pulling in a background thread to see if that flag has changed. But due to the nature of this design, messages would get lost/overwritten with false values when there was 2-way communication (i.e. Robot1 sending AND receiving messages from both Robot2 and HUB). Therefore, communication is *monogamous (*please reference image below). This means if Robot2 wants to talk to Robot1, it mus send that message to the Hub, which will then forward that message to Robot1.



4.2 Trial Failure

I end this document with the major roadblock that stumped me. The trial does not run into an error when in full autonomous mode and no error introduced. Additionally, the individual error handling code works smoothly. BUT, after an error is introduced and a troubleshooting option is chosen, the robot does not respond. I see that the robot is receiving the message from the Hub computer, but the 'flag' is not getting set, or the flag is being overwritten before the robot gets a chance to react. I've tried to extract the code that is responsible for moving the robot into an object that can be shared in 'locate' and 'communicator' (so when the robot receives the message it can move accordingly – circumventing the flag entirely) but an unusual syntax error is thrown by the args.h file for C files within the Player/Stage software.