Objective

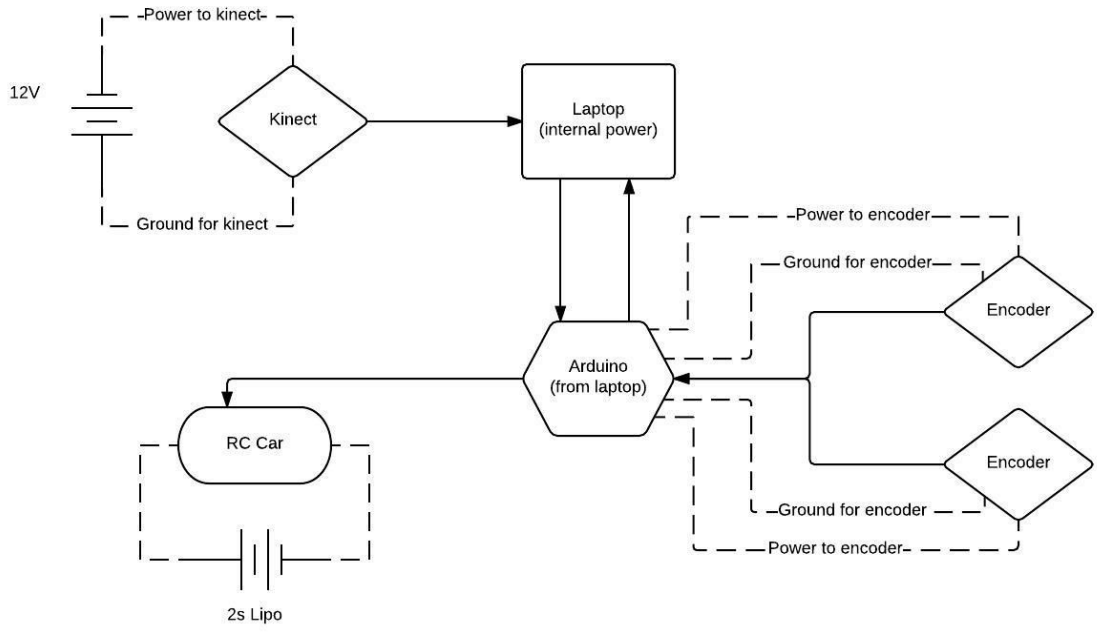
The objective of this project was to explore simultaneous localization and mapping techniques with an autonomous vehicle. We were to create a rover that could be placed into an unknown environment and be able to explore the environment while mapping its features.

The scope was to retrofit an RC car with wheel encoders and an Xbox 360 Kinect and integrate it with a laptop utilizing the Robot Operating System (ROS). In this project we explored the concepts of SLAM and everything that it would take to map and perform autonomous localization in an environment. There was also wireless internet control from another laptop enabling better remote operation.

SLAM implements a computer algorithm as a means of localizing a robot and mapping an unknown area. The current SLAM hardware includes sensors, motors and various hardware. Below are the bill of Materials, architecture and circuit schematic.

System

The system consisted of an RC car chassis with a motor and battery. The shock absorbers were replaced by solid struts, and a table shell was mounted to the car chassis on which to mount and sit the other additional components.



On this table, there are mounted two rotary quadrature encoders over the rear wheels which are wired to an arduino, which also controls the RC Car through its steering servo and remote receiver. The arduino sends encoder information to ROS on the laptop, and also receives instruction through its same connection. The laptop receives visual IR data from the kinect and has its own independent power source in the form of a 12V battery. The laptop provides power through the arduino to the encoders, steering servo, and remote receiver. The RC Motor also has its own battery. The table has components with various slots that fit together to allow for a variety of different size configurations as far as the wheel encoder placement and laptop size.

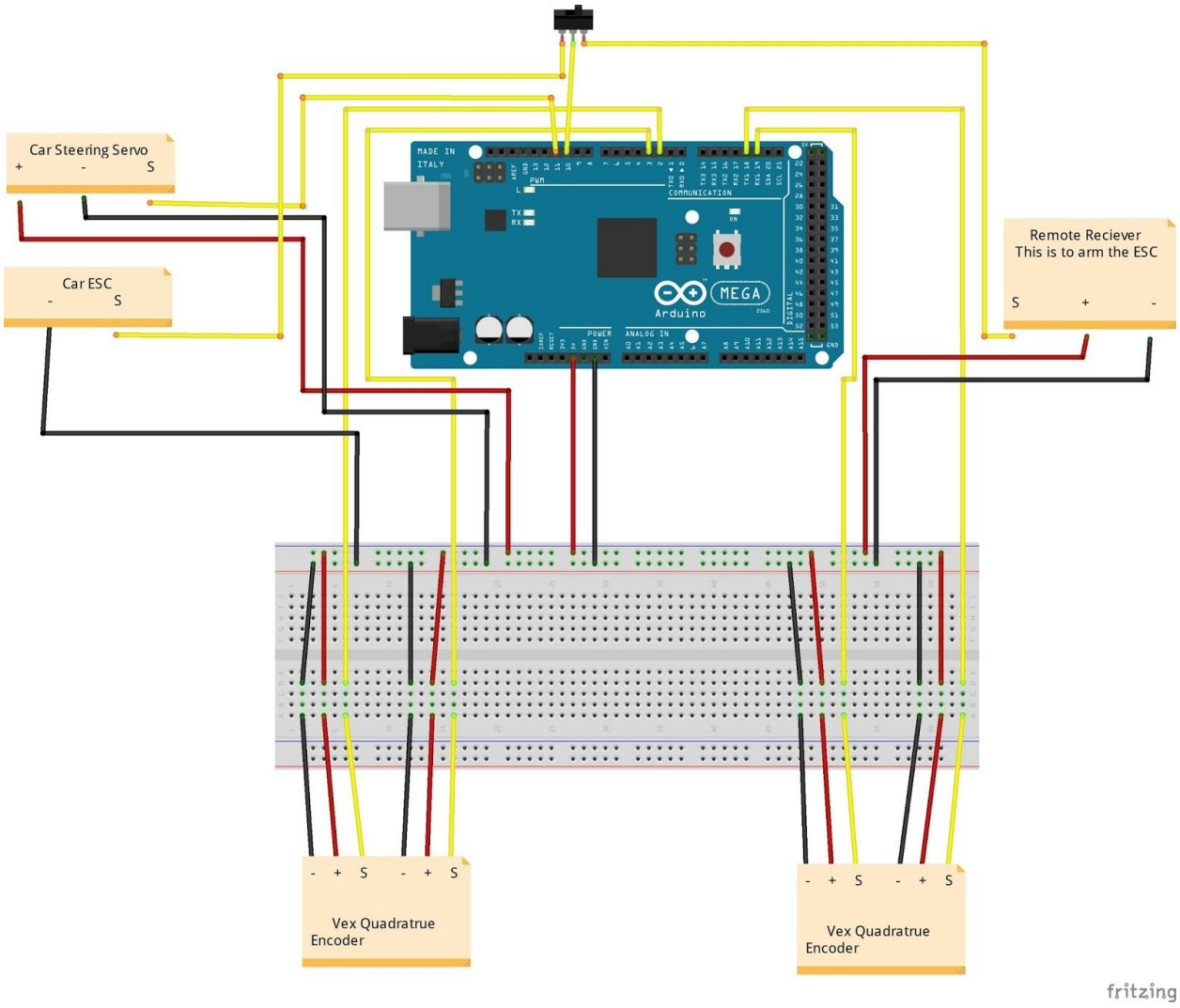
Parts List:

Description:

Price

RC Car	1/10 Vandal Quanum 4WD Electric Racing Buggy (RTR)	$128.62	Hobby King
Motor	Viper 21.5T 1500 kv brushless motor	Discontinued	
Replacement	LRP Vector K7 Sensored Brushless Motor (21.5T)	$62.99	Amain Hobbies
Battery (car)	Venom 20C 2S 3200mAh 7.4V LiPo Battery with Universal Plug	$28.62	Amazon.com
Kinect	X box 360 Kinect Sensor bar (used on amazon)	$40.00	Amazon.com
Battery	Rechargeable 3800mAh Lithium Ion w/ DC Connector, 12 volt	$24.99	Amazon.com
Arduino	ARDUINO MEGA 2560 REV3 (can find knock off cheaper)	$45.95	Arduino Store
Encoders	VEX Optical Shaft Encoder (2-pack)	$19.99	Robot Mesh
Total		$351.16	

Circuitry:



Usage Guide

The following operating system, program, packages and dependencies are needed in order to fully operate the project. The following needs to be install:

Operating System

- Ubuntu 14.04

Program

- ROS (Robot Operating System)

Packages within ROS

- Arduino
- Freenect
- Rtabmap

- Ros wall follow code

Dependencies

- Arduino wall follow sketch

Setup/Install

Go to http://wiki.ros.org/indigo/Installation/Ubuntu

-This is the current fully supported version
-Skip repositories section 1.1
-Make sure to get desktop full in 1.4 then go to 1.4 (This may take some time.)
-in 1.6 execute first to command boxes

- Check

Type roscore into terminal to see if ros is setup correctly

```
$ run source /opt/ros/indigo/setup.bash
```

-gives access to ros commands for indigo
-execute for every shell window

- Create A New Workspace

Follow the instructions on this website to create a workspace.
http://wiki.ros.org/catkin/Tutorials/create_a_workspace

- Arduino

Go to arduino webpage and download arduino for your bit rate of linux
-go to files and extract tar file “arduino nightly”
-in the terminal cd

```
$ cd arduino
$ chod +x install.sh
$ ./install.sh
$ sudo apt-get install ros-indigo-rosserial-arduino
$ sudo apt-get install ros-indigo-rosserial
```

Navigate in terminal to libraries folder in Arduino

```
$ rosrun rosserial_arduion make_libraries.py .
```

-makes all your library folders for arduino to ros (don;t forget the last period after py)

- Freenect

Navigate to src in catkin_ws

```
$ sudo apt-get install libfreenect-dev
$ sudo opt-get install ros-indigo-freenect-launch
$ cd..
$ catkin_make
$ git clone https://github.com/ros-drivers/freenect_stack.git
$ cd..
$ catkin_make
```

- Rtabmap

Go to catkin_ws/src in terminal

```
$ git clone https://github.com/introlab/rtabmap.git
$ cd rtabmap/build
$ cmake
$ make -j4
$ sudo make install
$ cd
```

```
$ rtabmap
```

See if it loads

If it loads, everything was installed correctly

- Arduino and ROS Wall Follow Code

Navigate to your documents directory

```
$ git clone https://github.com/neherh/Wall_Follow_Code.git
```

Once you git clond the file, place the arduino code into the Arduino libraries folder. Place the ROS wall_follow in catkin_ws/src directory and catkin_make the ROS code. Be aware, that the code was initially catknized for the Jade distribution and some modifications may be necessary.

Car Calibration

1. Open the wall_follow.yaml file found within the catkin_ws/src/wall_follow/launch directory.
2. Change the parameters accordingly. The following parameters need to be changed:

```
# Robot Dimensions
wall_follow/dimensions/tick_rev:       90
wall_follow/dimensions/dia_wheel:    0.087
wall_follow/dimensions/width_wheel:  0.215

# wall follow characteristics
wall_follow/wall_follow/des_wall_dist:          1
wall_follow/wall_follow/min_front_dist:         0.5
wall_follow/wall_follow/max_front_dist:         1.0
wall_follow/wall_follow/min_side_dist:          0.5
wall_follow/wall_follow/max_side_dist:          1.0
wall_follow/wall_follow/right_array_val:   12
wall_follow/wall_follow/center_array_val:  17

# controller
wall_follow/controller/kp:  0.9
```

Tick_rev is the resolution of the encoders: how many ticks per revolution.
Dia_wheel is the diameter of the wheels.
Width_wheel is the distance from wheel to wheel.

Change the wall follow characteristics as well as the controller as needed to get optimal wall following.

The right_array_val and centery_array_val correspond to the specific laserscan values that are chosen within the laserscan array; it would be best to not change those values unless major errors result from current values.

Car Operation

1. Check the wiring of the arduino by using the breadboard schematic illustrated above.
2. Check the workplace and remove any hazardous material.
3. Attach and make sure a tether to the rc car is secure. Attaching the tether ensures that if errors in the code occur, the car will not be damaged.
4. SSH from a different computer into the computer that is on the rc car (See Note 1 for examples and tutorials)
5. Power the the esc's and receiver. (The esc will need be armed manually, see Note 2 below on how to arm manually)
6. Once the esc is armed, lift the car up on a pedestal where the tires are free from any surface. Due to time constraints, extreme safety precautions were not taken, therefore once the program starts to run, the motor from the car will continuously run.
7. In new terminals, after the *source catkin_ws/devel/setup.bash* has occurred, run the following lines of code in 3 separate terminal windows to initialize wall_following:

```
$ roslaunch freenect_launch freenect.launch depth_registration:=true

$ roslaunch pointcloud_to_laserscan sample_node.launch

$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

The first line launches the kinect sensor, the second line launches the conversion from the camera images to laserscan data and the third initializes the ROS - to - Arduino communication. Before running the arduino communication, make sure that the port in the commandline is correct; see Note 3.

8. To listen to the topics use the following commandline:

```
$ rostopic echo topic
```

Replace the italicized *topic* with one of the following topics:

- camera/scan
- wall_follow/odometry_topic
- wall_follow/traj_angle

The topics show the laserscan data, odometry data from the encoders and the trajectory angle in radians that the ROS publishes to the arduino.

Monitoring the trajectory topic as well as the terminal where the arduino is ran from is a good idea; see Note 4.

9. To initiate mapping run each line of the following code on separate terminals

```
$ roslaunch rtabmap_ros rgbd_mapping.launch rtabmap_args:="--delete_db_on_start"
```

The above code launches rtabmap and begins mapping.

10. When safe to do so, place the rc car on the floor to begin mapping
11. Once finished, press ctrl-c to close each individual terminals safely.

NOTES

Note 1: Examples and Tutorials to SSH

- https://www.pluralsight.com/blog/software-development/how-to-install-and-use-ssh-secure-shell-in-ubuntu
- http://www.wikihow.com/Check-the-IP-Address-in-Linux

In order to ssh into another computer, make sure that the internet connection of the computer on the rc car and the computer being operated both have the same internet **AND** the internet is not secure. Ie. if you see an internet address that looks something similar to this 10.35.154.179, with the 10 at the front, then that is a secured network and you will not be able to ssh into the computer.

Note 2: Manually arming the esc controller

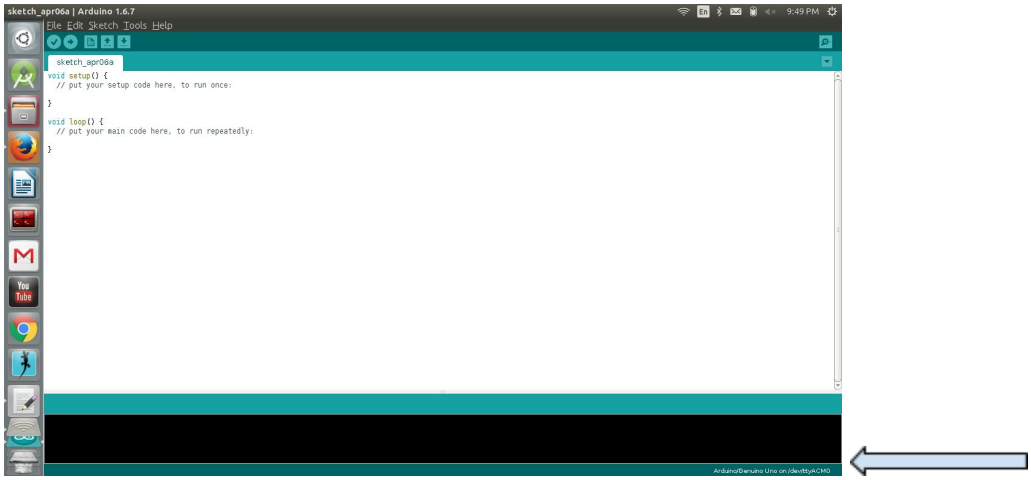
To arm manually, remove the signal wire (yellow wire from the esc) and connect to the receiver signal pin with the remote controller turned on and wait for the esc to be armed. The connection can be on any signal pin on the receiver. To determine if the esc is armed, the esc will output a musical chiming sound. Once the chiming sound is heard, remove the signal pin from the esc and place it back to it's normal position as indicated in the schematic.

Note 3: Arduino Commandline and port verification

To run the communication between the Arduino and a computer via ROS, the following command must be used (or it's variant commands):

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

The port that is specified in the commandline is ttyACM0 (zero at the end). To determine what port your arduino is on, go into an Arduino sketch and verify the port by plugging the arduino into the computer and looking on the bottom right portion of your screen.



Blown up, the port call looks like this on Ubuntu:

Some examples of different ports are:

- /dev/ttyUSB0
- /dev/ttyUSB1
- /dev/ttyACM0
- /dev/ttyACM1

Use the above information about your arduino and replace the */dev/ttyACM0* with the correct port name.

Note 4: Monitoring the Arduino terminal

Monitoring the Arduino terminal is imperative because at times the serial communication between the Arduino and ROS on the computer is disconnected or the buffer is saturated and the communication becomes corrupted. When the serial communication is corrupted, warning messages will be outputted on the Arduino terminal. When the messages are outputted, be aware that the car may not run or the motors may go haywire, so caution must be used.