SLAM Developers Kit

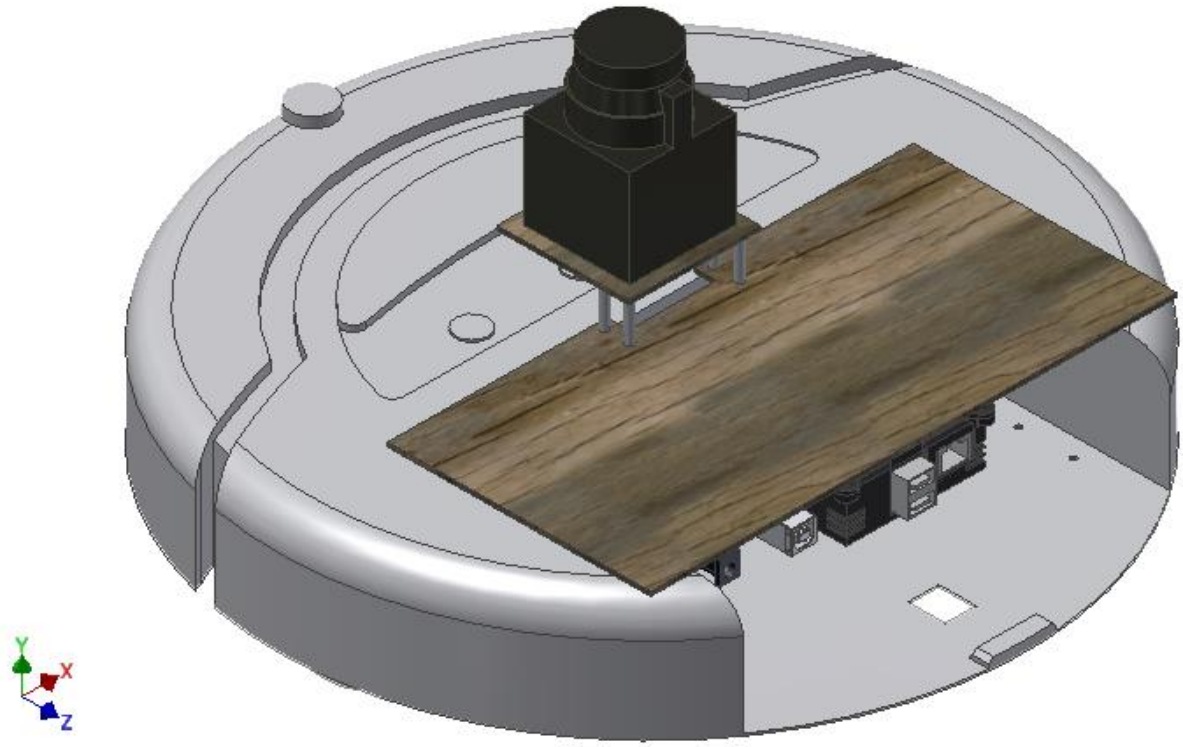
User Manual

By: Eysteinn Finnson, Ólafur Þórðarson &

Sindri Ólafsson

Introduction

The SLAM developer's kit is a simple mobile robot platform and is aimed at absolute beginners in robotics. In about one hour anyone with very basic programing knowledge should have an up and running robot and an example code to start playing with.



What is included?

HOKUYO urg-04lx LIDAR:	1200$
iRobot Create:	130$
Raspberry pi, empty SD card:	40$
Arduino UNO:	20$
Wires and connectors:	30$
LiPo 2200mAh:	30$
dc-dc converter:	20$
support structure	20$
A5020 optical flow sensor	10$
Total cost:	1500$

Setting up the Raspberry

1. Install an operating system on your Raspberry (Raspbian tested but any linux system should work). Step 1 refers to the official Raspberry Pi documentation: http://www.raspberrypi.org/documentation/installation/installing-images/
2. Connect the Raspberry to a monitor and the internet.
3. Log in to the Raspberry. Default username is 'pi' and password is 'raspberry'.
4. Open terminal and update the system "$ sudo apt-get update".
5. Create a suitable directory for the program. "$ mkdir SLAM && cd SLAM".
6. Clone the git repository to get source code "$ git clone https://github.com/ruSLAM/ruSLAM.git".
7. Now you have all the files needed to run the included modules.

Setting up the Arduino

1. Fetch the Arduino IDE http://arduino.cc/en/Main/Software
2. Wire the Arduino as showed on figure 2.
3. Find Arduino_settings.ino in ruSLAM source code.
4. Upload it to the Arduino

Configuring USB settings

1. When Both the LIDAR and Arduino have been connected to the Raspberry via USB, unplug the LIDAR and wait for 10 seconds.
2. Plug it back in and write "$ dmesg" in Raspberry's terminal.
3. Observe the last lines of the output and look for ttyUSB0, ttyACM0 or similar. This is the serial port name assigned to the LIDAR.
4. Open LIDAR.cpp and change lines 16 and 17 if needed to the serial port name found in step 3
5. Do the same for the Arduino and if needed change lines 5 and 6 in ArduinoCom.cpp.

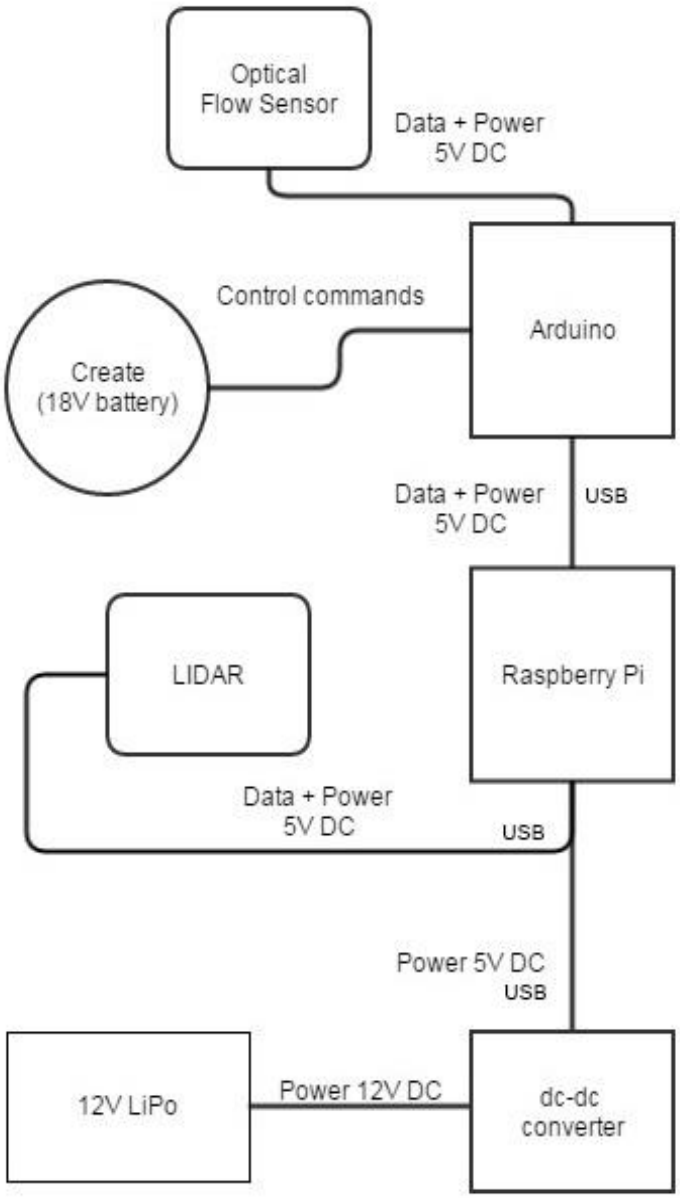
Permanently fixing the Baud rates is a headache so we will go for a simpler solution

1. Run "$ stty -F /dev/ttyACM0 115200" for the LIDAR. (use your serial port name)
2. Run "$ stty -F /dev/ttyACM1 9600" for the Arduino. (use your serial port name)
3. Run "$ stty -F /dev/ttyACM0 –a" to check if everything is ok. (use your serial port name)

   In some cases the stty command will not synchronize the LIDAR properly. Another way to set the baud rates for the LIDAR is using mgetty.

1. "$ sudo apt-get install mgetty"
2. "$ sudo mgetty -s 115200 /dev/ttyACM0" let this run for about 3 seconds
3. [ctrl + C]
4. "$ stty -F /dev/ttyACM0 –a"

Wiring the system



The LIDAR must be as close to the center of rotation as possible see figure 3. It's elevation however does not matter as much.

The power source for everything except the Create is a 3 cell LiPo battery connected to a 12 to 5 V dc-dc converter. These converters are widely available as car 12V USB chargers. It must be able to supply at least 1.5A. Over voltage and current protection are a good quality.

Create's battery can be used as a power source. Pins 10, 11 and 12 on the connector supply 18V battery power up to 1.5A. A dc-dc converter that converts $18V \pm 2V$ to regulated USB 5V can be connected to these pins to provide power to the entire system.

The optical flow sensor must hover closely above the ground and be rested on a lever arm see figure 4. Adjusting the parameter that describes this lever arm might be needed. Default value is 112.5 mm.

Figure 1 This is the system diagram. Note the split USB cable between the dc-dc converter, LIDAR and Raspberry Pi. This is done because the LIDAR Draws more current than the raspberry can provide. The Raspberry is also powered through this cable.

Connecting Arduino to A5020 and Create

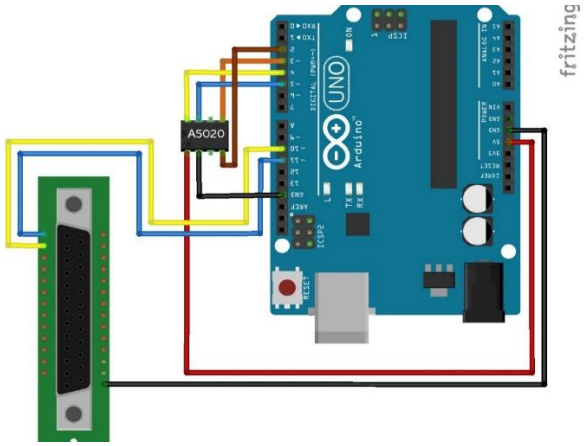


Figure 2 Connecting the Arduino to the Create and optical flow sensor. USB port connects to the Raspberry.

The A5020 optical flow sensor can be harvested from a certain DELL mouse. It might be possible to use another sensor but it is not guaranteed. A driver for the A5020 optical flow sensor and instructions on how to connect it can be found online.

> URL: https://strofoland.wordpress.com/2013/08/08/reading-a5020-optical-sensor-using-arduino-part2/
> POSTED BY: STEFANPIRVU
> POSTED ON: AUGUST 8, 2013

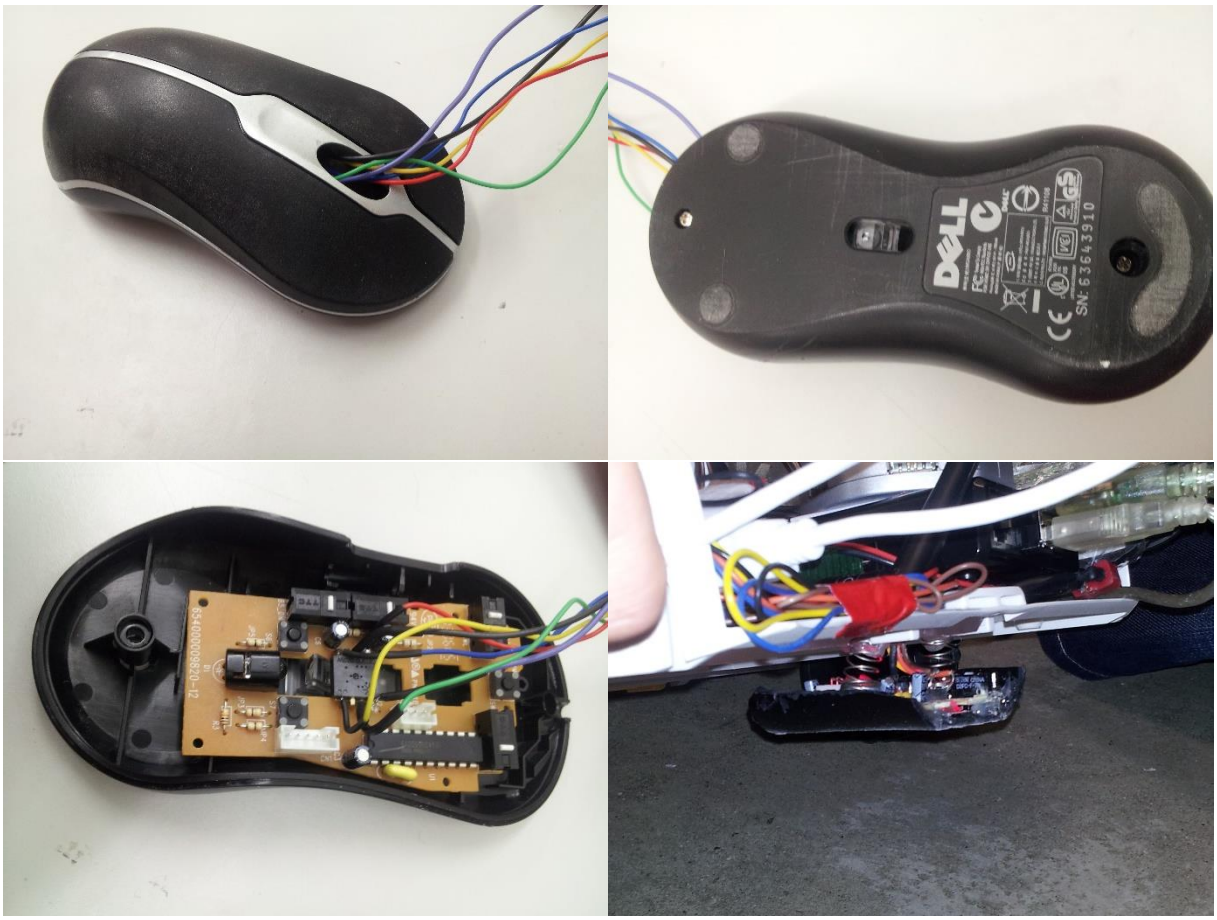


Figure 3 The dell mouse used for this project. Dissected, hot glued to springs and placed under the create on a lever arm see figure 4

Odometry Explained

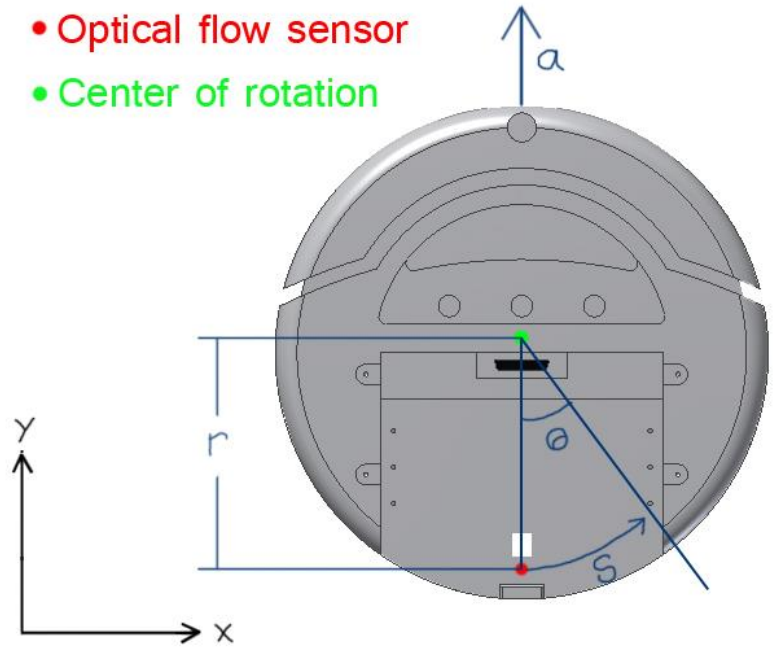


Figure 4 iRobot Create's axis and dimensions represented by letters.

The optical flow sensor can't track rotation by itself so to measure the rotation it must be at the end of a lever arm. Now the sensor can't turn on the spot so it's able to record all movement as either forward or sideways. To calculate the yaw of our setup simple trigonometry is applied. $d\theta = \frac{ds}{r}$ This approach assumes the sensor's resolution is much greater than the lever arm r and instantaneous movement ds appears to be sideways translation only. Forward translation a and angle θ are converted to x and y coordinates. CCW rotation returns positive s and forward translation returns positive a so the calculations are as follows.

$$dx = -da \cdot \sin(\theta)$$

$$dy = da \cdot \cos(\theta)$$

Finally the instantaneous changes are summed up to calculate position from an initial reference point (0,0) $xpos = \sum dx$, $ypos = \sum dy$ and $\theta = \sum d\theta$. All of these calculations are done in real time on the Arduino. The raspberry then requests the xpos, ypos and θ coordinates as needed.

Command library for Roomba

Arduino command	Meaning	Corresponding Raspberry function
DFW	Drive Forward	Void Roomba.DriveForward()
DBW	Drive Backwards	Void Roomba.DriveBack()
STP	Stop	Void Roomba.Stop()
TNR	Turn Right until instructed otherwise	
TNL	Turn left until instructed otherwise	
TLX***	Turn left for specified number of degrees f.ex. TLX090	void Roomba. TurnLeft(int angle)
TRX***	Turn right for specified number of degrees f.ex. TLX090	void Roomba. TurnRight(int angle)
RST	Reset all variables	void Roomba.ResetPos()

Command library for Odometry

Arduino command	Meaning	Corresponding Raspberry function
XYA	Get X coordinates Y coordinates and Angle in the form “#x#y#a”	string Roomba.GetPos()
	After running GetPos this function will return the X value as an integer.	int Roomba.Xpos();
	After running GetPos this function will return the Y value as an integer.	int Roomba.Ypos();
	After running GetPos this function will return the Angle value as a double.	double Roomba.Angle();

Command library for LIDAR

Arduino command (Not relevant)	Meaning	Corresponding Raspberry function
	Starts the HOKUYO LIDAR	void Radar.start();
	Flush LIDAR buffer	void Radar.flushLidar()
	Request one scan of the surroundings. Only for HOKUYO LIDAR	void Radar.getSwipe()
	Gets the raw data from the HOKUYO LIDAR	string Radar.getData()
	Decodes the raw data from HOKUYO LIDAR and returns it in mm	void Radar.Decoder(int Data[], string Data_string)

Plotting the data from example.cpp

The data is exported to a file 'Datalog.txt' in the format seen in the right column below. New Data signifies a new data point where location and LIDAR sweep are extracted.

Flag:	New Data
Xpos [mm]:	0
Ypos [mm]:	0
Angle [rad]:	0
Lidar Data 683 lines [mm]:	652
	512
	420
	…
Flag:	New Data
Xpos: [mm]	20
…	…

Plotting this data is relatively straight forward using Octave or Matlab. The yaw of the Roomba must be taken in to consideration when converting LIDAR Data to Cartesian coordinates. This code only covers one sweep.

```
theta = linspace(-2.0944,2.0944,683);        % Range of the LIDAR defined
rad = Lidar_Data(5:688)                      % All 683 data points collected in one sweep

for i = 1:683
  cx(i) = cos(theta(i) - yaw) * rad(i) + xpos    % yaw is the same as Angle
  cy(i) = sin(theta(i) - yaw) * rad(i) + ypos
end

scatter(cx, cy, 5, r, 'filled')
```

This way to visualize data is by no means optimal or recommended. However seeing things work in such raw manner is good for understanding the basics. We leave the job of particle filtering and writing occupancy grid mapping algorithms to the user. ROS has some good tools for this that can be installed on the Raspberry.