Self Driving Bot

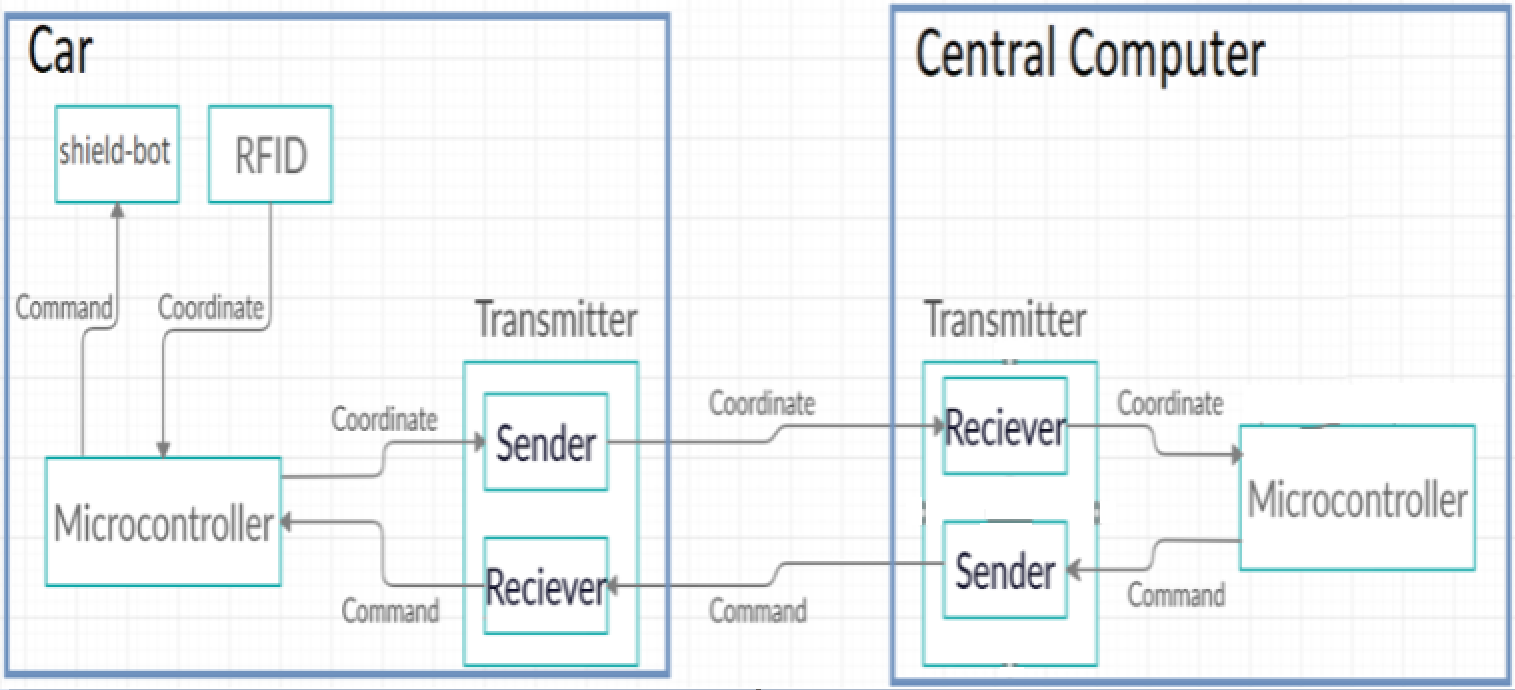
By: Aryan Rashidi-Tabrizi

# Description

This is a basic controller for a visualization of the location of a Self Driving Bot. This bot will follow black lines on the ground using the IR sensors.

For the purpose of designing this specific application, the following design was proposed.  As it can be seen in figure below, in this design, as the RFID reader(connected to the car) scans the RFID tag, the id of the scanned tag is passed to the transceiver. Afterward, the transceiver sends the id of the tag to the server. In return the server responds with a specific command (depending on the id of the tag sent). The transceiver receives the command(reply) from the server and passes it to the bot.

This project also demonstrate how to use DigitalInput, DigitalOutput, AnalogInput, and PwmOutput.



The Seeed Shield bot data sheet is available here: <http://wiki.seeedstudio.com/Shield_Bot_V1.2/>. It has the following available ports:

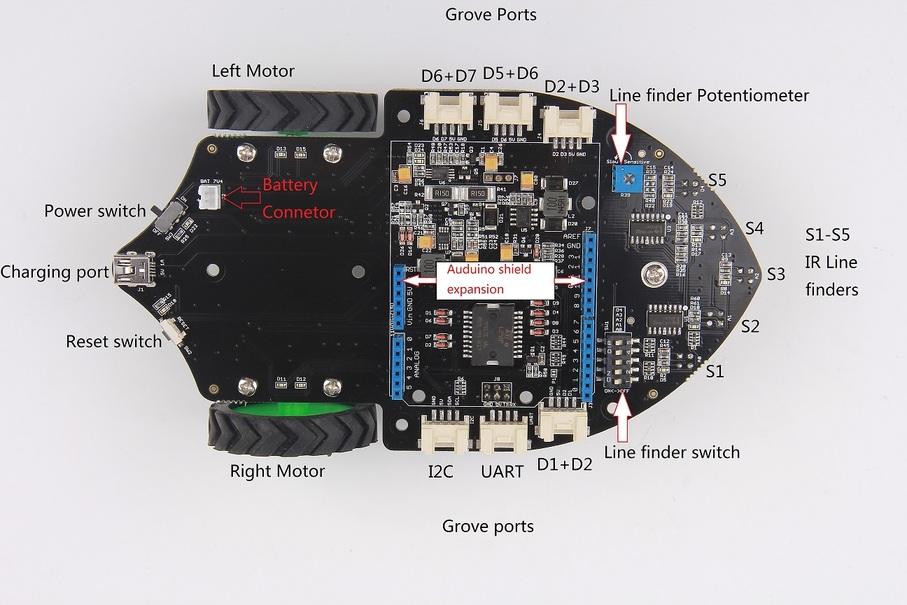


Figure 1: Seeed Bot Component Diagram

The light sensor used is the Grove Light Sensor, it should be connected to the I2C port shown in the above diagram. This works because the I2C pins are multiplexed with the analog output pins. For more information about this sensorsee the grove documentation: <https://www.seeedstudio.com/Grove-Light-Sensor-v1-2.html>. The table below contains the pins used and what they are connected too.

|  |  |
| --- | --- |
| Pin | Connection |
| A0 | Right IR |
| A2 | Center IR |
| A5 | Light Sensor |
| D4 | Left IR |
| D8 | Right Motor 1 |
| D9 | Right Motor Enable |
| D10 | Left Motor Enable |
| D11 | Right Motor 2 |
| D12 | Left Motor 1 |
| D13 | Left Motor 2 |

Table 1: Pin Connection Table

# Block Diagram

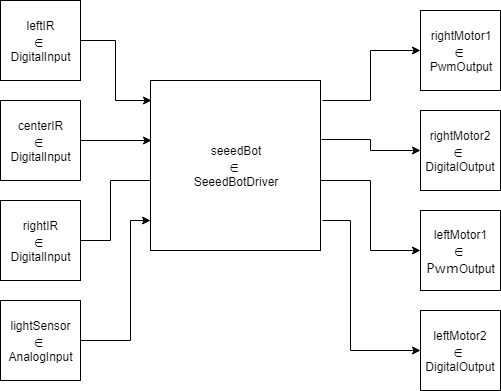


Figure 2: Model Coupling Diagram

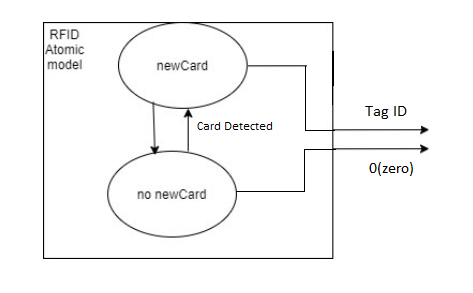


Figure 3: RFID Atomic Model

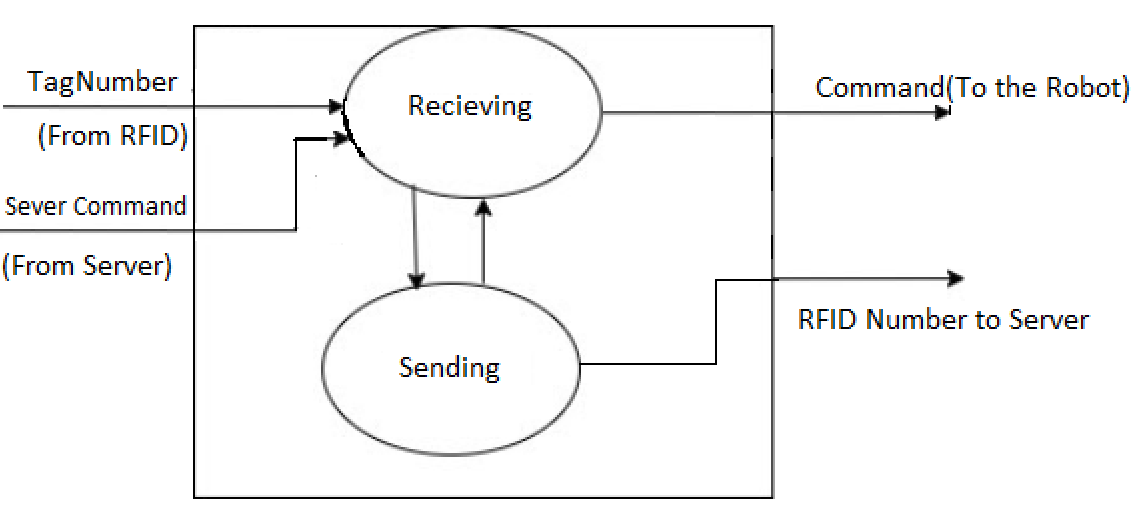


Figure 4: Tranmitter Atomic Model

# Formal Models

ATOMIC: SeeedBotDriver

This model will evaluate the IR sensors and the light sensor’s input and decide which direction to steer the car. Using the direction decided, the output function will drive the motors.

**State Variables:**  
**sigma** = , **phase** = Passive;

**leftIR** = false  bool;

**centerIR** = false  bool;

**rightIR** = false  bool;

**light** = 0  float;

**dir** = unknown { right, straight, left, stop, unknown }

**Formal specification:**

**X** = { leftIR  boolean, centerIR  boolean, rightIR  boolean, lightSensor  float } /\* Used to receive input from the GPIO models \*/

**Y** = { rightMotor1  float, rightMotor2  boolean, leftMotor1  float, leftMotor2  boolean } /\* Used to control the motor speed and direction\*/

**S** = { phase, sigma, leftIR, centerIR, rightIR, light, dir, prop }

**ext** ( s = { phase, sigma, leftIR, centerIR, rightIR, lightSensor, dir, prop}, e, x = { leftIR, centerIR, rightIR, light } ) {

oldDir = state.dir;

case x

leftIR:

state.leftIR = x;

centerIR:

state.centerIR = x;

rightIR:

state.leftIR = x;

lightSensor:

state.light = x

if ( state.leftIR + state.rightIR + state.centerIR > 1 || state.light < 0.3 ) {

/\* Direction unclear, stop the bot \*/

State.dir = stop;

} else if (state.rightIR) {

state.dir = left;

} else if (state.leftIR) {

state.dir = right;

} else {

state.dir = straight;

}

If (oldDir != dir) {

State.prop = true;

}

}

**int** (s= { phase, sigma, leftIR, centerIR, rightIR, light, dir, prop }) {

State.prop = false;

}

****s) {

switch(state.dir)

case right:

rightMotorOut1 = 0.5;

rightMotorOut2 = 0;

leftMotorOut1 = 1;

leftMotorOut2 = 1;

case left:

rightMotorOut1 = 1;

rightMotorOut2 = 1;

leftMotorOut1 = 0.5;

leftMotorOut2 = 0;

case straight:

rightMotorOut1 = 0.5;

rightMotorOut2 = 0;

leftMotorOut1 = 0.5;

leftMotorOut2 = 0;

case stop:

default:

rightMotorOut1 = 0;

rightMotorOut2 = 0;

leftMotorOut1 = 0;

leftMotorOut2 = 0;

send rightMotorOut1, rightMotorOut2, leftMotorOut1, and leftMotorOut2 to the output ports;

}

COUPLED: Top

This is the top model, notice that it does not have any inputs or outputs. All embedded projects should not have I/O in their top models. This is because GPI/O is done inside atomic models, such as the DigitalInput and DigitalOutput models.

X = {};

Y = {};

D = {

leftIR  DigitalInput, rightIR  DigitalInput, centerIR  DigitalInput,

lightSensor  AnalogInput, seeedBot  SeeedBotDriver, rightMotor1  PwmOutput,

rightMotor2  DigitalOutput, leftMotor1  PwmOutput, leftMotor1  DigitalOutput

};

EIC = {};

EOC = {};

IC = {

( leftIR.out, seeedBot.leftIR ), ( centerIR.out, seeedBot.centerIR ), ( rightIR.out, seeedBot.rightIR ),

( lightSensor.out, seeedBot.lightSensor ), ( seeedBot.rightMotor1, rightmotor1.in ),

( seeedBot.rightMotor2, rightmotor2.in ), ( seeedBot.leftMotor1, leftMotor1.in ),

( seeedBot.leftMotor2, leftMotor2.in )

};