Final Hack: Remote Control

1 Authors

Aravind Kumaraguru, Scout Heid

2 Description

This hack enables a fully wireless remote control of the robot using cheap Nordic radios operating at 2.4 GHz. The robot will be controlled via a breadboard with two potentiometers that control the speed of the left and right motors.

3 Materials

Bill of Materials

Component	Quantity	Unit Price	Vendor
MSP430-G2553	2	$9.99	Ti.com
Nordic NRF24L01	2	$1.50	eBay.com
Breadboard (400 tie point)	2	Free	EE40 Lab
9V 48:1 gearmotors	2	$4.29	robotshop.com
66mm Tamiya wheels	2	$3.20	robotshop.com
LM1086 Vreg	2	Free	EE40 Lab
LM6484 Quad Op Amp	1	Free	EE40 Lab
BC547B BJT	2	Free	EE40 Lab
10K potentiometers	2	$0.95	Adafruit.com
Soft T18 Knobs	2	$0.50	Adafruit.com
Resistor (10K Ohm)	4	Free	EE40 Lab
Resistor (20K Ohm)	2	Free	EE40 Lab
Capacitor (10 uF)	4	Free	EE40 Lab
Diode	2	Free	EE40 Lab
9V battery	2	Free	EE40 Lab
9V battery connector	2	Free	EE40 Lab
Female-to-Male jumpers	24	Free	EE40 Lab

Shopping List

Component	Quantity	Price
MSP430-G2553	1	$9.99
Nordic NRF24L01	2	$3.00
9V gearmotors	2	$8.58
66mm wheels	2	$6.40
10K potentiometers	2	$1.90
Soft T18 Knobs	2	$1.00
TOTAL		**$30.87**

Alternate Shopping List

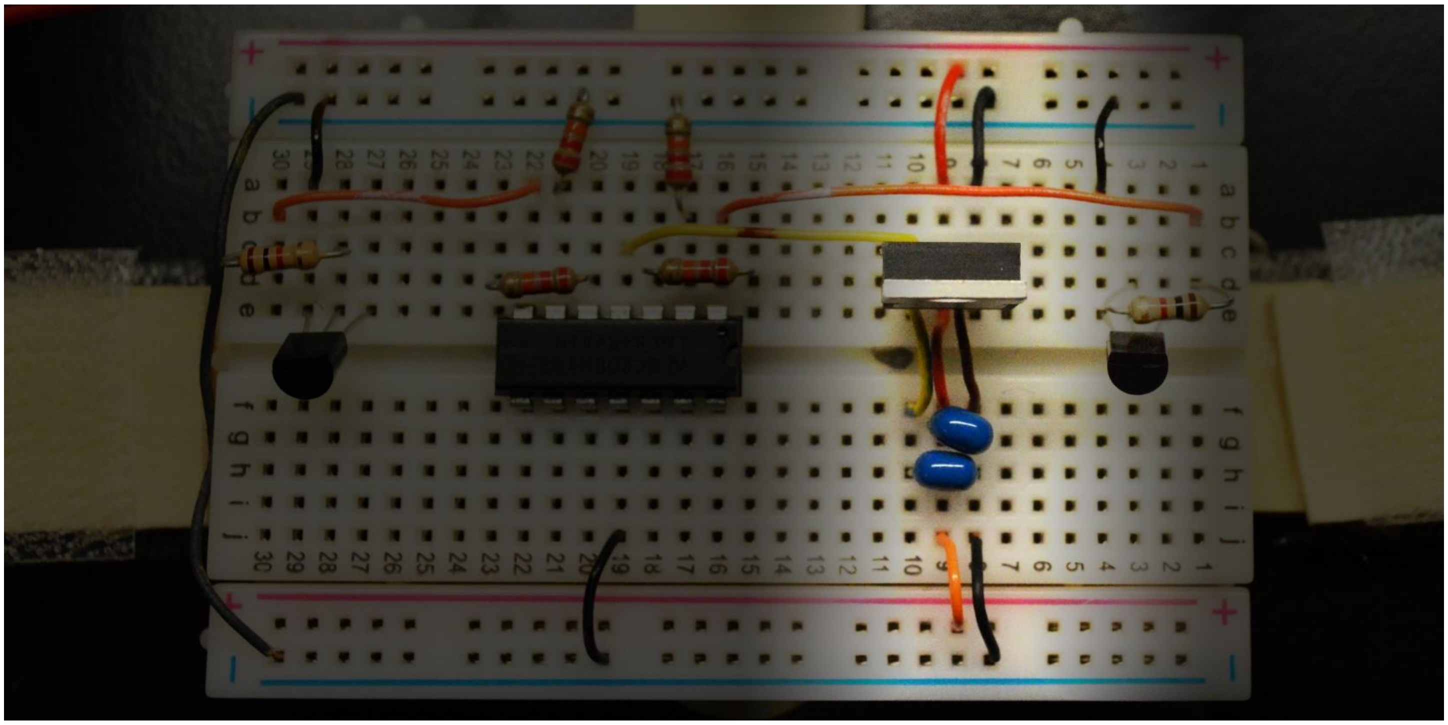
Component	Quantity	Price
MSP430-G2553	1	$9.99
Nordic NRF24L01	2	$3.00
10K potentiometers	2	$1.90
Soft T18 Knobs	2	$1.00
Springs	2	FREE
TOTAL		**$15.89**

For a smaller budget, use the given motors with eccentric weights and springs.

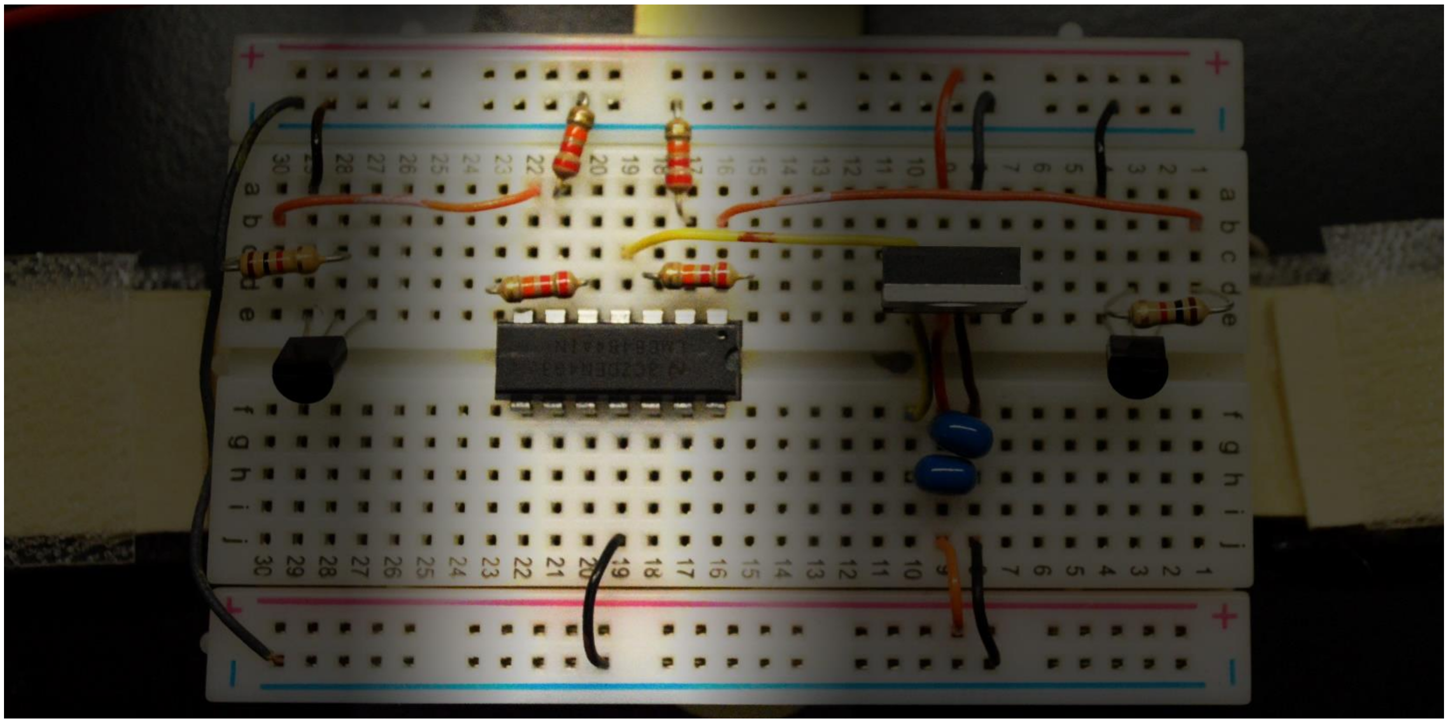
4 Instructions

4.1 Build robot

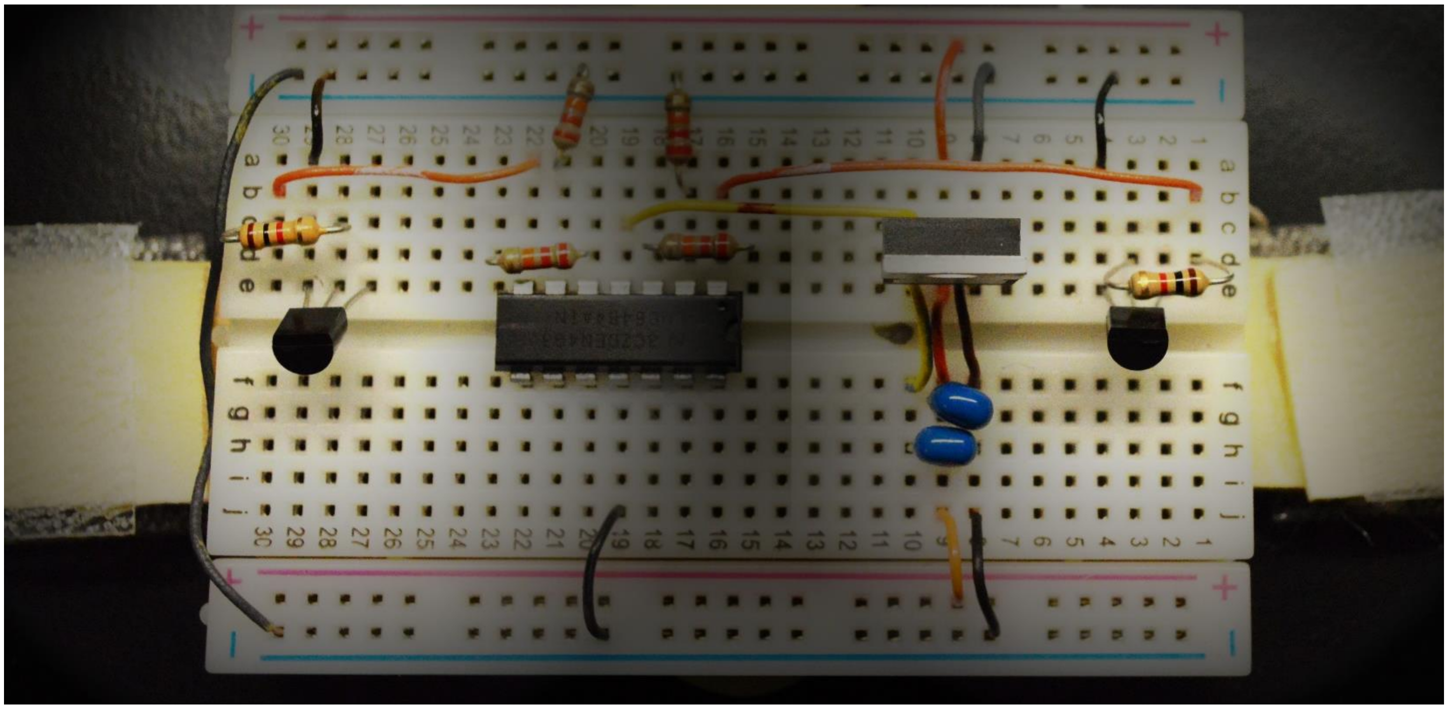
Connect voltage regulator subcircuit



Connect Op Amp subcircuit – provides buffering and voltage gain for the MSP430 PWM signals. Be sure to supply 9V to the Op Amps positive power rail, not 3.3V. [Datasheet for LMC6484]



Connect BJT subcircuit – provides the current gain needed to drive the motors


Ignore the resistor attached to the base of each BJT: Those were removed to improved performance. Refer to schematic for more detail!

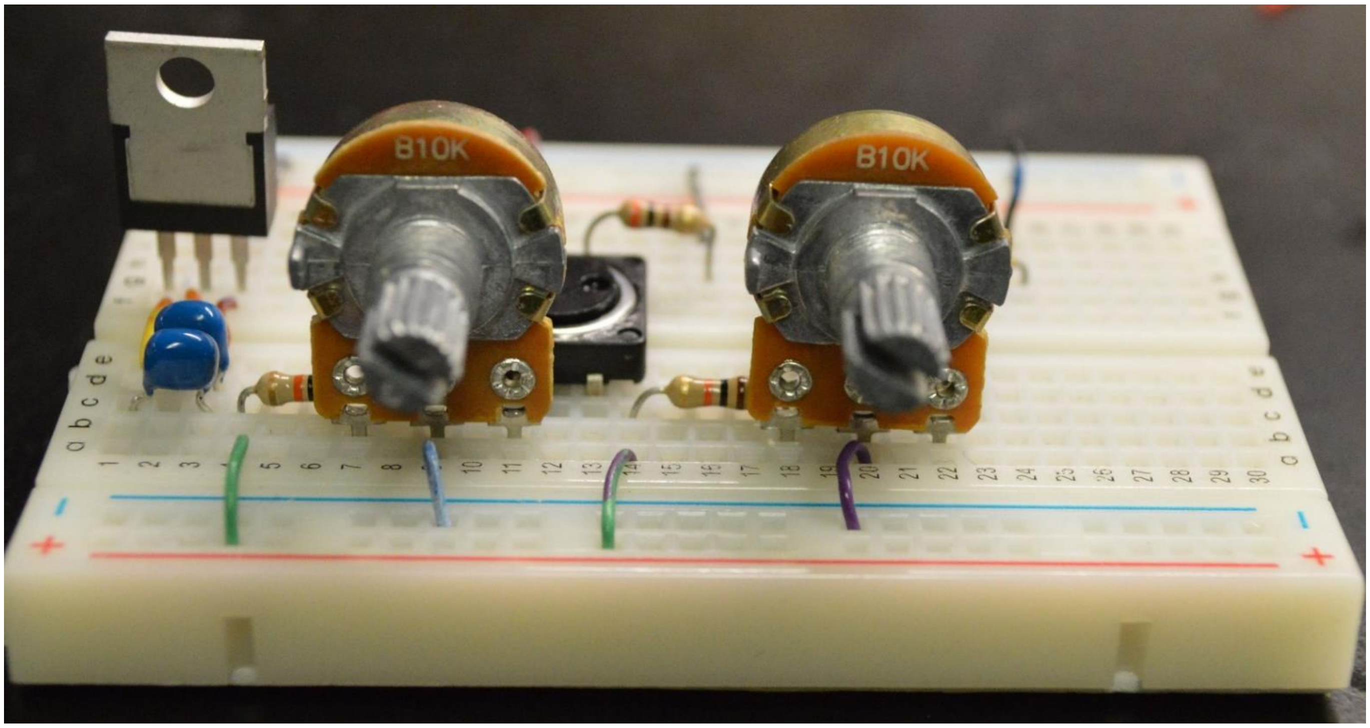
Clip any connector on the leads of the motors and strip a small amount of insulation to that the leads can be inserted into a breadboard. Solder on the diodes in parallel as specified by the schematic.

Also, build a suitable robot frame onto which you can mount a breadboard, 2 motors, a 9V battery, and an MSP430.

4.2 Build remote control

Connect voltage regulator subcircuit – same idea as the vreg subcircuit on the robot's breadboard

Connect potentiometer subcircuit – a voltage divider with high resistance to minimize current draw

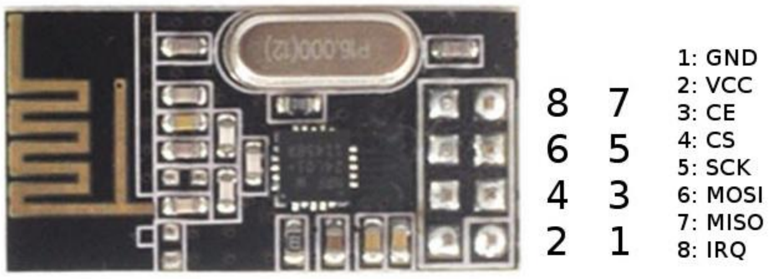


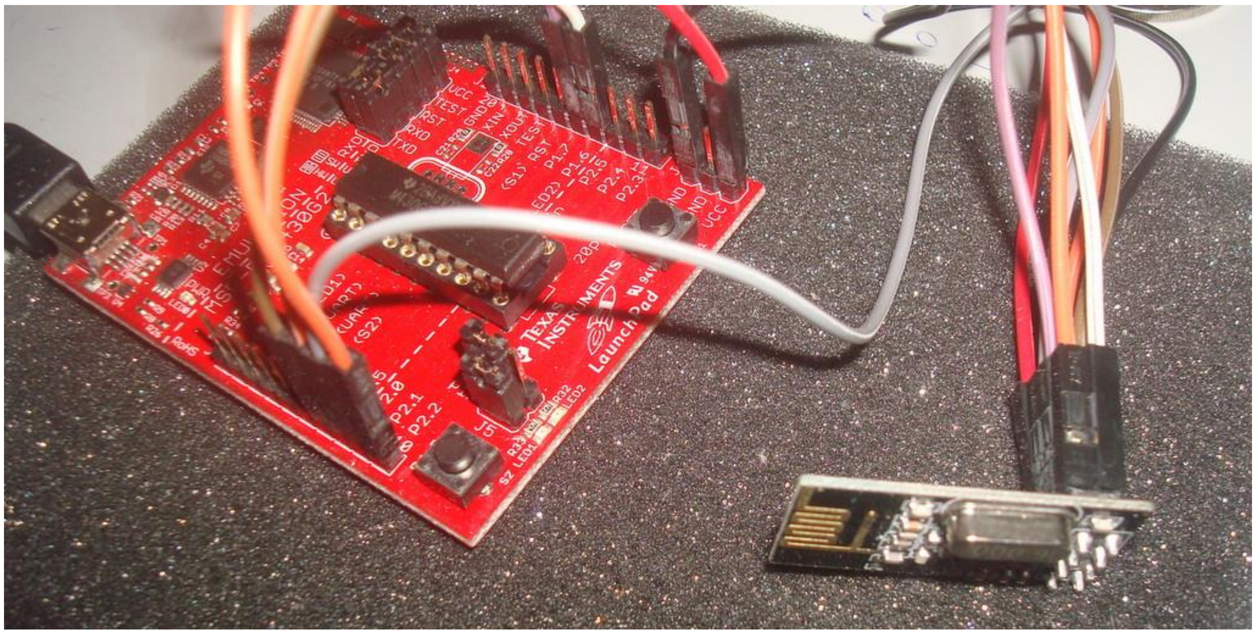
Ignore the button in this picture – that is from an alternate design we scrapped earlier.

4.3 Connect radios

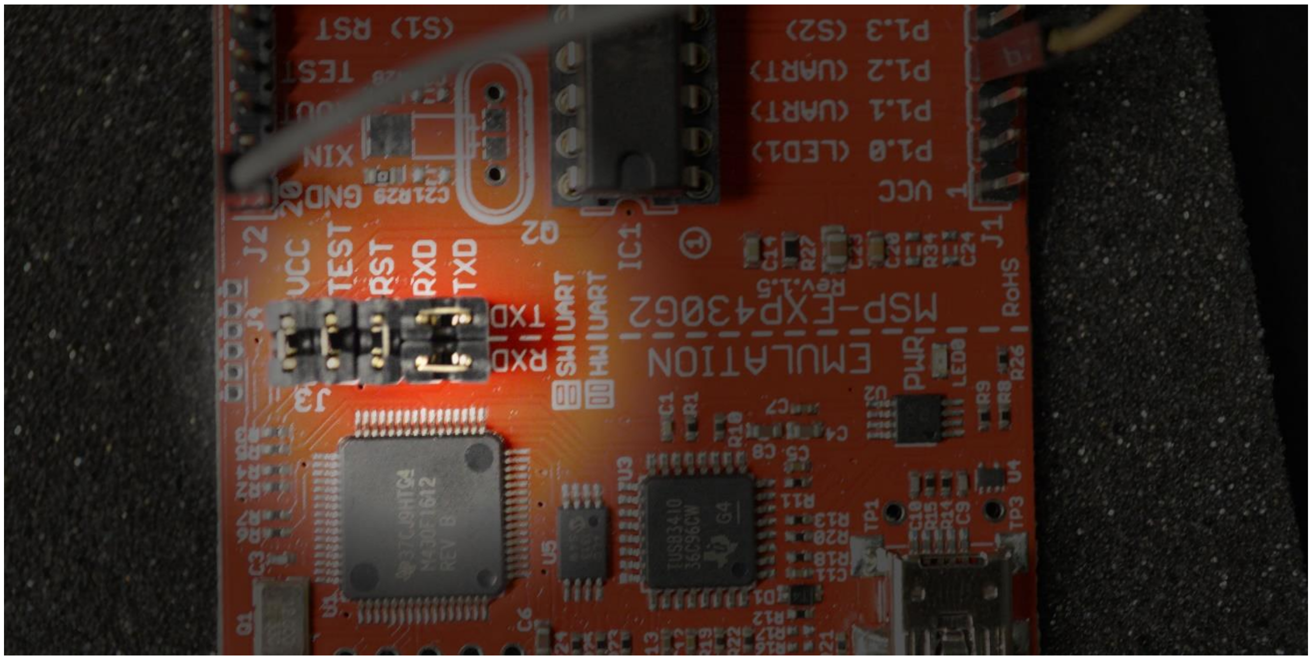
Connect each Nordic Radio to an MSP430 with female-to-male jumper wires. Follow the schematic and this pin layout diagrams. Labelling your jumpers makes it easier to keep track of the pins.

P1.5: SCK
P1.6: MISO
P1.7: MOSI
P2.0: CE
P2.1: CSN
P2.2: IRQ
VCC: VCC
GND: GND





Also, be sure to reorient the RXD and TXD jumpers on both MSP430's so that they look like this:



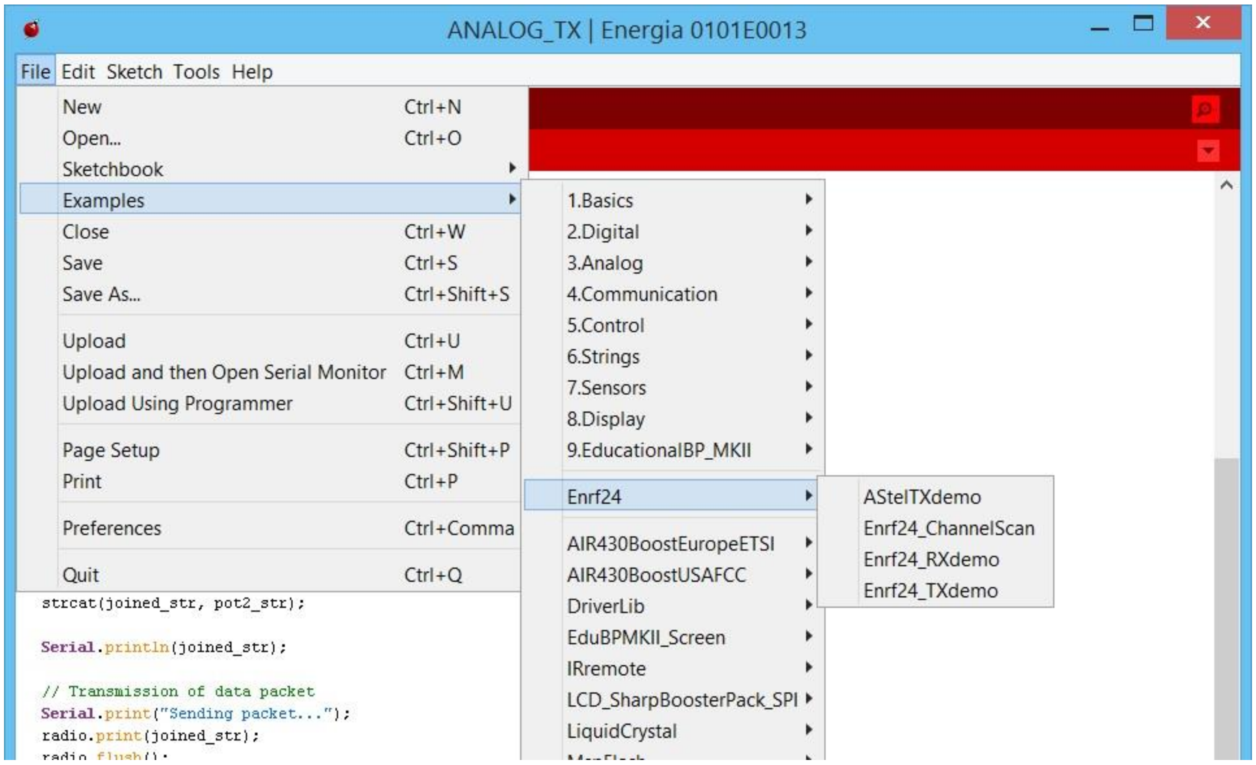
4.4 Set up Development Environment

The code for this hack will not work with Code Composer. The Energia IDE must be used. Information on how to set up this software can easily be found online.

Download and set up Energia: http://energia.nu/download/

Download and set up the nRF24L01+ library for Energia: http://energia.nu/reference/libraries/

The example code for running the Nordic radios should now be available.



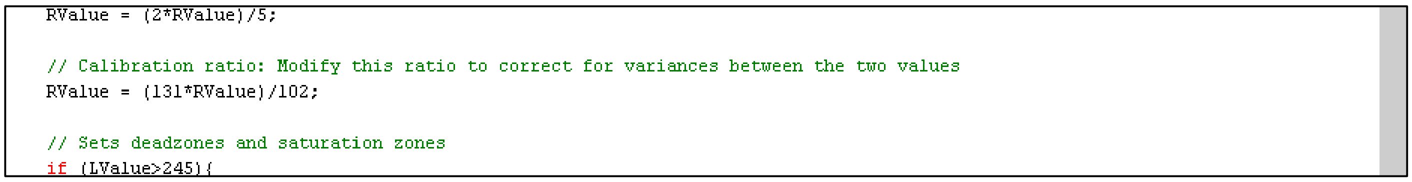
```
strcat(joined_str, pot2_str);

Serial.println(joined_str);

// Transmission of data packet
Serial.print("Sending packet...");
radio.print(joined_str);
radio.flush();
```

4.5 Code and Testing

Load Enrf24_TXdemo onto one MSP430 and Enrf24_RXdemo onto the other. If the setup was successful, the red led on the TX controller will blink. The MSPs' status can also be verified via the serial monitor in Eneriga.

Next, load ANALOG_TX onto one MSP430 and ANALOG_RX onto the other. The TX controller will connect to the remote control breadboard, and the RX will connect to the robot. Refer to the schematic to determine which pins to connect where.

Calibration: The response from the potentiometers may slightly vary, creating a constant gap between the two readings even when the potentiometers are tuned to the same angle. This can be compensated for in the RX code:



```
RValue = (2*RValue)/5;

// Calibration ratio: Modify this ratio to correct for variances between the two values
RValue = (131*RValue)/102;

// Sets deadzones and saturation zones
if (LValue>245){
```

Watching the serial print statements, tweak the operation in this line so that RValue and LValue match up.

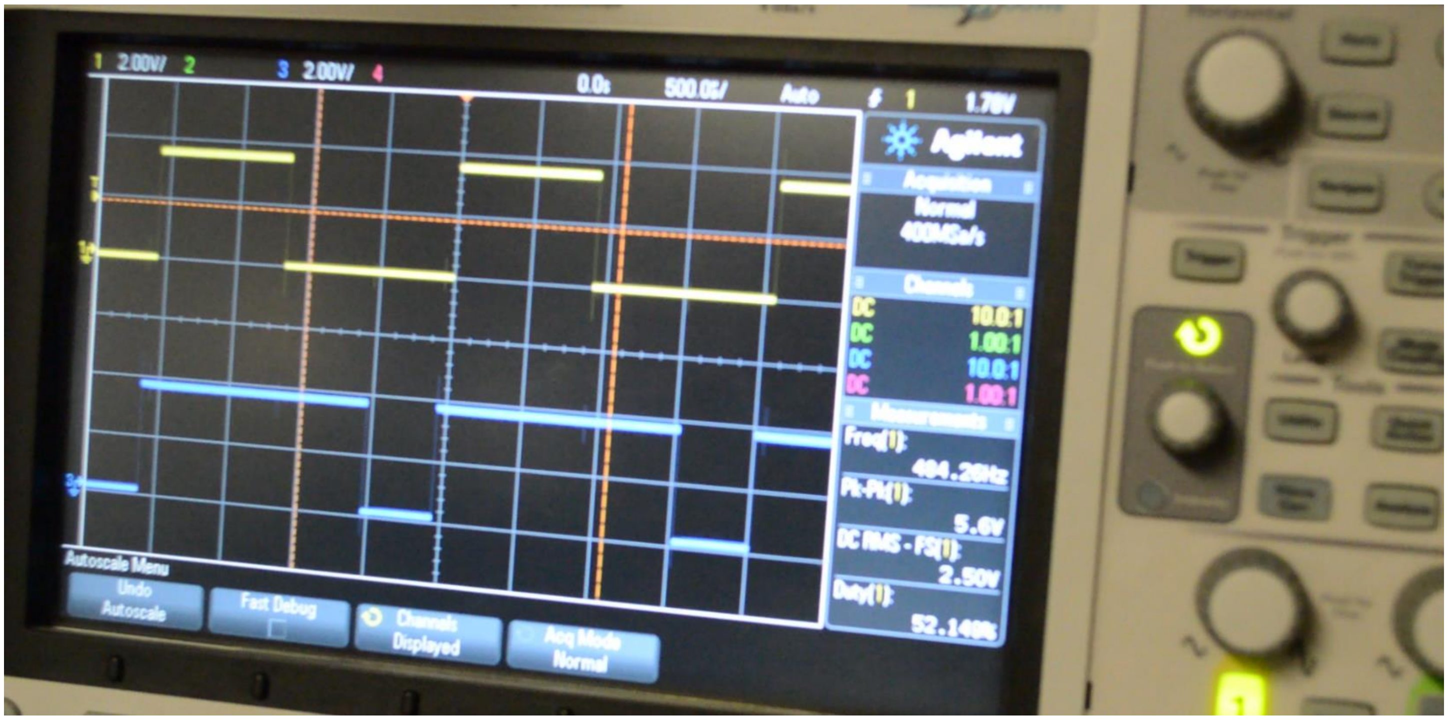
4.6 Finalize Hack

Attach the breadboards, motors, and microcontrollers together. Read the schematic carefully. Have fun!

5 Documentation

PWM signal on oscilloscope

http://youtu.be/xF9sg3VZHmM



Test Drive

http://youtu.be/e0LC6fBKWEA

6 Code

Code can be downloaded from: https://github.com/AravindK95/EE40_Remote_Control

6.1 ANALOG_TX

```
#include <Enrf24.h>
#include <nRF24L01.h>
#include <string.h>
#include <SPI.h>


char pot1_str[4];
char pot2_str[4];
char joined_str[8];

Enrf24 radio(P2_0, P2_1, P2_2);  // P2.0=CE, P2.1=CSN, P2.2=IRQ
const uint8_t txaddr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x01 };
void dump_radio_status_to_serialport(uint8_t);


void setup() {

  Serial.begin(9600);
  Serial.println("Running Setup...");

  pinMode(P1_1, INPUT);
  pinMode(P1_3, INPUT);

  // Initializes serial interface with radio
  SPI.begin();
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);

  radio.begin();  // Defaults 1Mbps, channel 0, max TX power
  dump_radio_status_to_serialport(radio.radioState());
  radio.setTXaddress((void*)txaddr);

}

void loop() {

  Serial.print("Building packet: ");

  // Reads input values and stores the results in character arrays
  itoa(analogRead(P1_1), pot1_str, 10);
  itoa(analogRead(P1_3), pot2_str, 10);

  strcat(joined_str, pot1_str);
  strcat(joined_str, ",");
  strcat(joined_str, pot2_str);

  Serial.println(joined_str);

  // Transmission of data packet
  Serial.print("Sending packet...");
  radio.print(joined_str);
  radio.flush();
  dump_radio_status_to_serialport(radio.radioState());

  // Flush memory for next cycle
  Serial.println("Flushing data...");
  strcpy(pot1_str, "");
  strcpy(pot2_str, "");
  strcpy(joined_str, "");

}

void dump_radio_status_to_serialport(uint8_t status)
{
  Serial.print("Enrf24 radio transceiver status: ");
  switch (status) {
    case ENRF24_STATE_NOTPRESENT:
      Serial.println("NO TRANSCEIVER PRESENT");
      break;

    case ENRF24_STATE_DEEPSLEEP:
      Serial.println("DEEP SLEEP <1uA power consumption");
      break;

    case ENRF24_STATE_IDLE:
      Serial.println("IDLE module powered up w/ oscillators running");
      break;

    case ENRF24_STATE_PTX:
      Serial.println("Actively Transmitting");
      break;

    case ENRF24_STATE_PRX:
      Serial.println("Receive Mode");
      break;

    default:
      Serial.println("UNKNOWN STATUS CODE");
  }
}
```

6.2 ANALOG_RX

```
#include <Enrf24.h>
#include <nRF24L01.h>
#include <string.h>
#include <SPI.h>

Enrf24 radio(P2_0, P2_1, P2_2);
const uint8_t rxaddr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x01 };
void dump_radio_status_to_serialport(uint8_t);


char* LStr;
char* RStr;
int LValue;
int RValue;
char inbuf[33];


void setup() {
  Serial.begin(9600);
  Serial.println("Running Setup...");

  pinMode(P1_2, OUTPUT);
  pinMode(P2_5, OUTPUT);

  // Initializes serial interface with radio
  SPI.begin();
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);

  radio.begin();  // Defaults 1Mbps, channel 0, max TX power
  dump_radio_status_to_serialport(radio.radioState());

  radio.setRXaddress((void*)rxaddr);
  radio.enableRX();  // Start listening
}

void loop() {

  dump_radio_status_to_serialport(radio.radioState());  // Should show Receive Mode

  // Loop freezes when radio not available
  while (!radio.available(true))
    ;
  if (radio.read(inbuf)) {
    Serial.print("Received packet: ");
    Serial.println(inbuf);

    // String deconstruction, type conversion and value adjustment
    LStr = strtok (inbuf ,",");
    LValue = atoi(LStr);
    LValue = (2*LValue)/5;


    RStr = strtok (NULL, ",");
    RValue = atoi(RStr);
    RValue = (2*RValue)/5;

    // Calibration ratio: Modify this ratio to correct for variances between the two values
    RValue = (131*RValue)/102;

    // Sets deadzones
    if (LValue<15){
      LValue = 5;
    }
    if (RValue<15){
      RValue = 5;
    }

    // Update PWM signals
    analogWrite(P1_2,LValue);
    analogWrite(P2_5,RValue);
    Serial.println(LValue);
    Serial.println(RValue);

  }
}

void dump_radio_status_to_serialport(uint8_t status)
{
  Serial.print("Enrf24 radio transceiver status: ");
  switch (status) {
    case ENRF24_STATE_NOTPRESENT:
      Serial.println("NO TRANSCEIVER PRESENT");
      break;

    case ENRF24_STATE_DEEPSLEEP:
      Serial.println("DEEP SLEEP <1uA power consumption");
      break;

    case ENRF24_STATE_IDLE:
      Serial.println("IDLE module powered up w/ oscillators running");
      break;

    case ENRF24_STATE_PTX:
      Serial.println("Actively Transmitting");
      break;

    case ENRF24_STATE_PRX:
      Serial.println("Receive Mode");
      break;

    default:
      Serial.println("UNKNOWN STATUS CODE");
  }
}
```