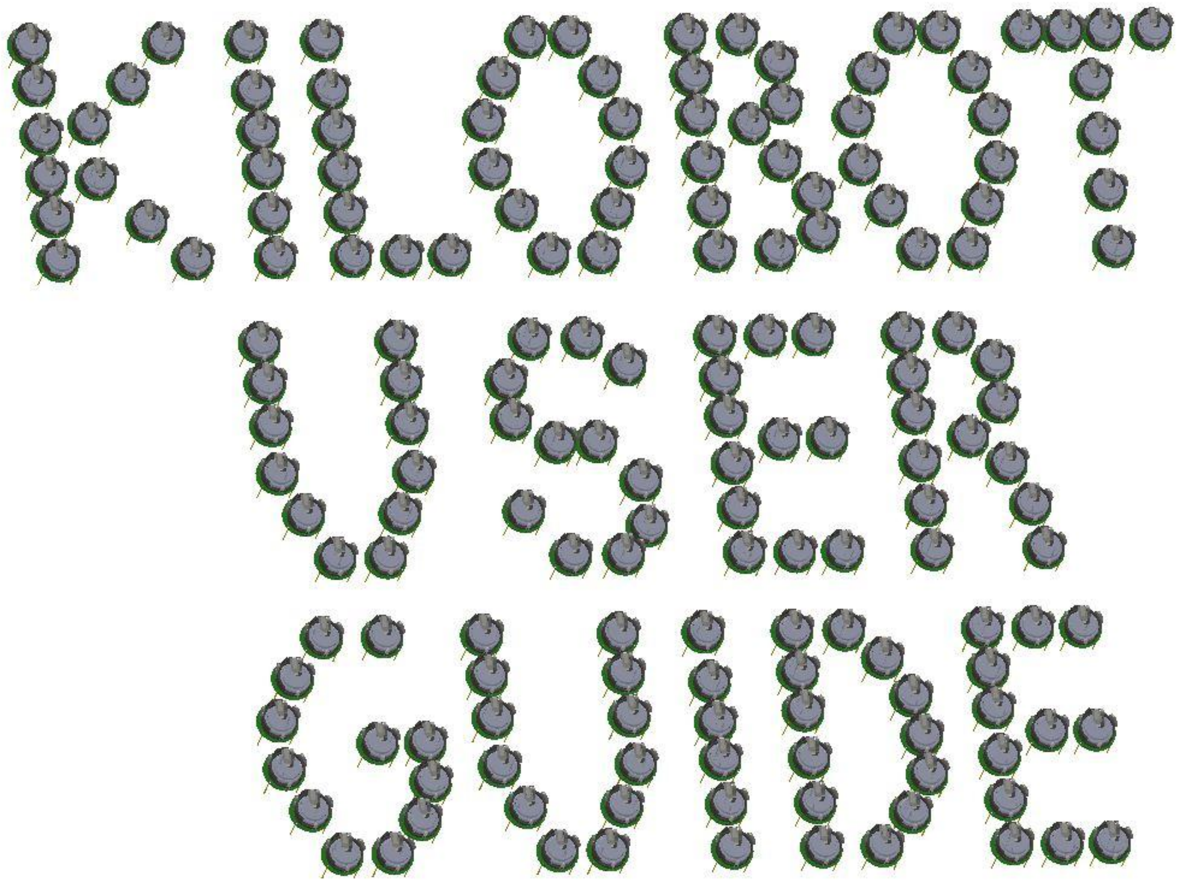


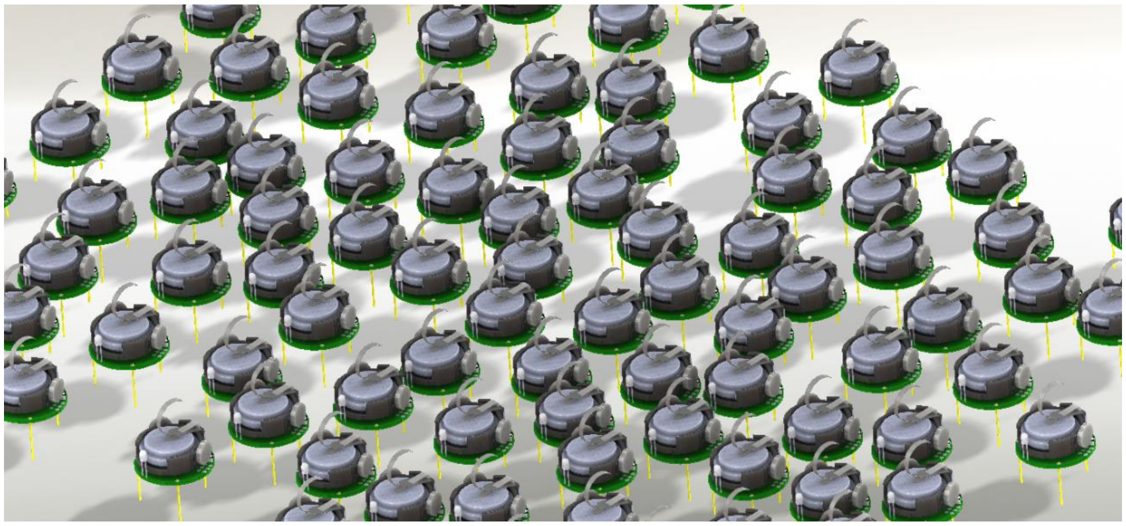
Beta Release

Harvard University

Self-Organizing Systems Research Group

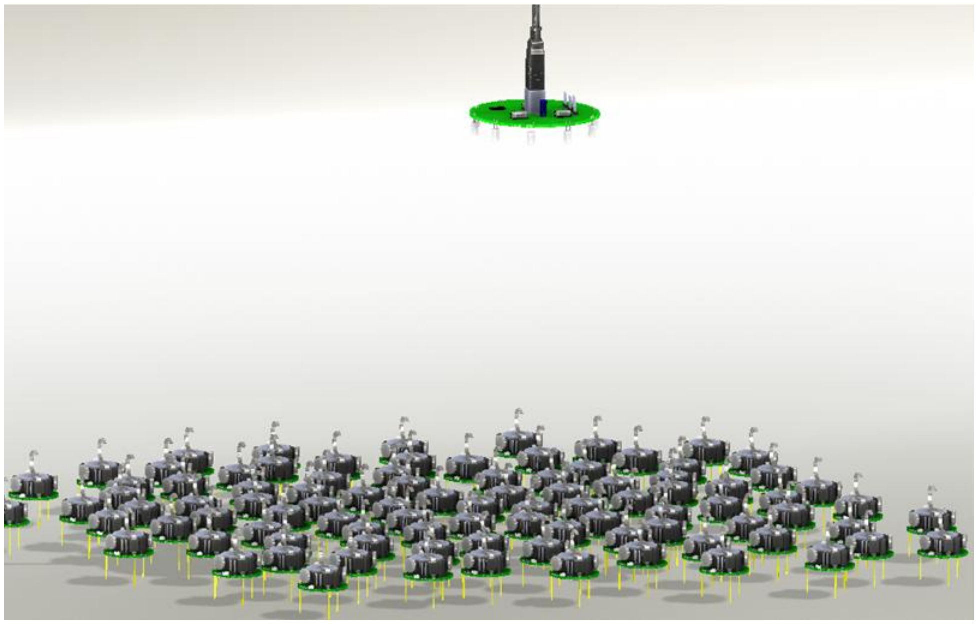
http://www.eecs.harvard.edu/ssr/

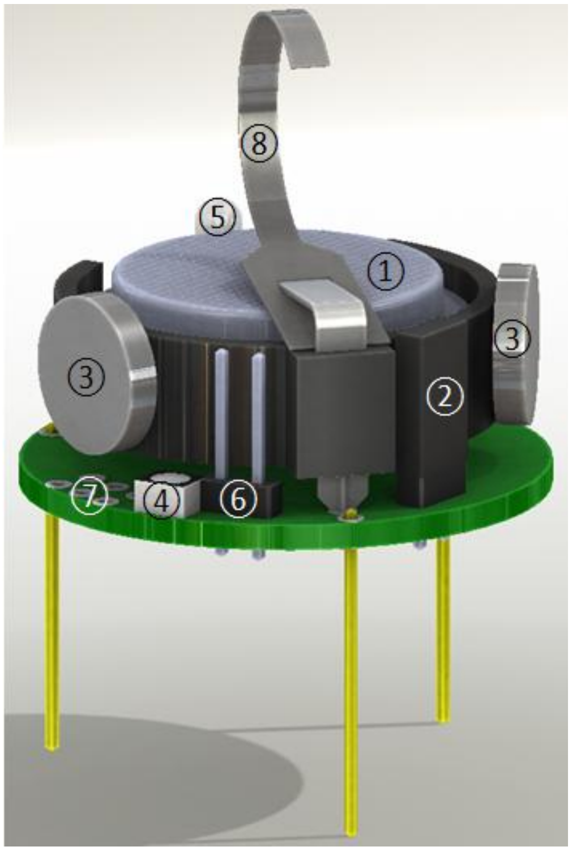
Kilobot System Overview

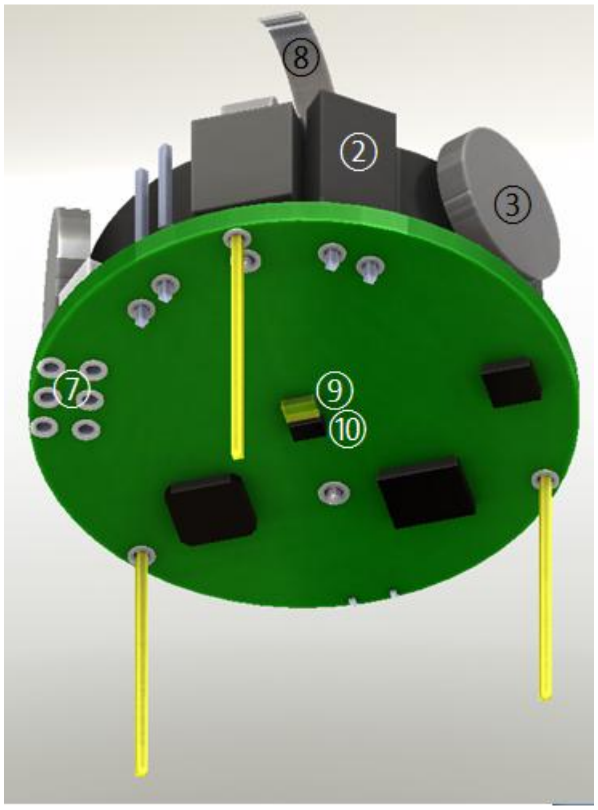


Kilobots are low cost robots designed at Harvard University's Self-Organizing Systems Research Lab http://www.eecs.harvard.edu/ssr/ . The robots are designed to make testing collective algorithms on hundreds or thousands of robots accessible to robotics researchers. Though the Kilobots are low-cost, they maintain abilities similar to other collective robots. These abilities include differential drive locomotion, on-board computation power, neighbor-to-neighbor communication, neighbor-to neighbor distance sensing, and ambient light sensing. Additionally they are designed to operate such that no robot requires any individual attention by a human operator. This makes controlling a group of Kilobots easy, whether there are 10 or 1000 in the group.

The Kilobot system consists of the Kilobot robots, an over-head controller (OHC) used to program and control the robots, an arena for operating the robots, and a charging station. In addition a windows computer is required to connect to the OHC via USB.







① 3.7-Volt Battery

② Power jumper

③ Vibration motors

④ LED (Red/Green/Blue)

⑤ Ambient light sensor

⑥ Serial output header

⑦ Direct programming socket

⑧ Charging Tab

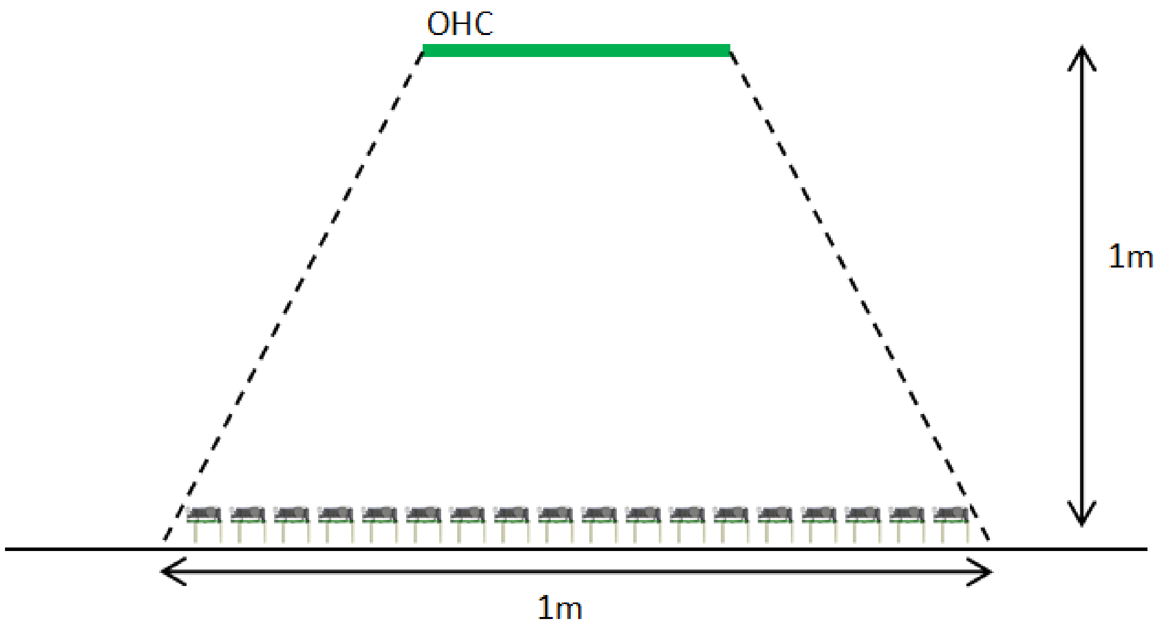
⑨ IR Transmitter

⑩ IR Receiver

System Setup

Kilobots should be operated on a smooth, flat, level surface to ensure proper robot mobility. To aid communication, the surface should be glossy or reflective. A dry-erase whiteboard oriented horizontally is recommended. To prevent communication interference, Kilobots should be operated in a location out of direct sunlight or other bright sources of Infra red light.

The overhead controller should be hung above the Kilobots at a distance of about one meter. The robots beneath the OHC in a about a one meter diameter region will be able to receive messages from the OHC as shown.



System Features

Kilobots

- Communication
    - Kilobots can communicate with neighbors up to 7 cm away by reflecting infrared (IR) light off the ground surface.
- Sensing
    - When receiving a message, distance to the transmitting Kilobot can be determined using received signal strength.
    - The brightness of the ambient light shining on a Kilobot can be detected.
    - A Kilobot can sense its own battery voltage.
- Movement
    - Each Kilobot has 2 vibration motors, which are independently controllable, allowing for differential drive of the robot.
    - Each motor can be set to 255 different power levels
- Each Kilobot has a onboard microcontroller (atmega 328)
    - 32K program memory (used for both user program and bootloader)
    - 1K EEPROM for storing calibration values and other non-volatile data.
- Each Kilobot has a built-in charger, which charges the onboard battery when +6 volts is applied to any of the legs, and GND is applied to the charging tab.
- Each Kilobot has an red/green/blue (RGB) LED pointed upward, and each color has 3 levels of brightness control.
- A serial output header is available on each robot for debugging via computer terminal.

OverHead Controller (OHC)

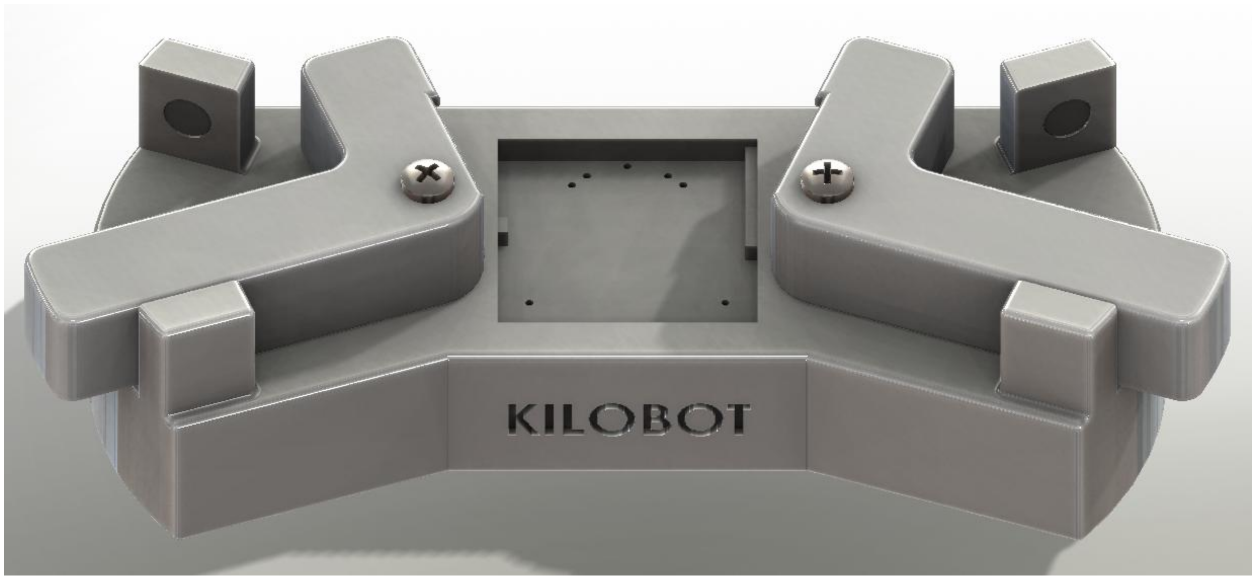
- The OHC can communicate with Kilobots below using infrared light allowing a user to:
    - send a new program to all Kilobots at once.
    - control the Kilobots such as pausing and powering on/off
- The OHC can program an individual Kilobot using the programming cable.
- The OHC can connect to an individual Kilobot's serial output header so it can be displayed on a windows computer.

Building the Kilobot System Hardware

To build a functional Kilobot system you will need to build 1) the robots, 2) the OHC, 3) a robot calibration board, and 4) a charging station. This section covers the physical building and assembly of these components.

If you have purchased your robots, skip to the **Software Installation Guide** section.

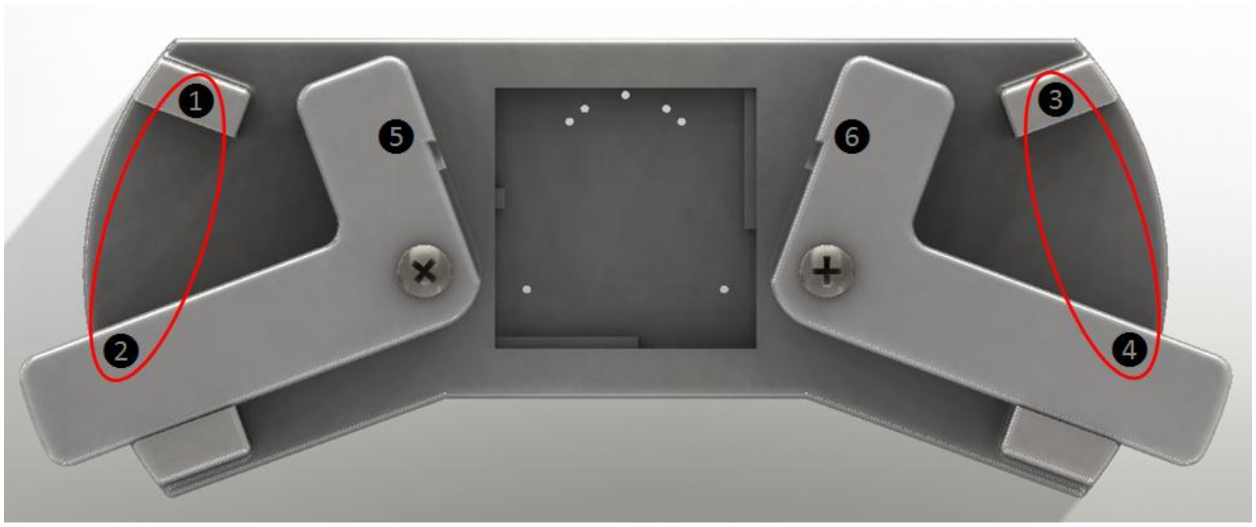
When building the robots it will be helpful (but not necessary) to use the robot assembly jig shown below.



This jig helps hold the robot legs and the power and debug headers for quick and precise soldering. It also holds the motors for quick and precise gluing to the robot. The specification for the assembly jig can be found on the web.

Building Assembly Jig

1. Produce jig parts from the files, available on the web.
2. Superglue magnets into jig arms. Make sure magnets 1 and 2 are placed so that they attract each other. Similarly make sure magnets 3 and 4 are placed so that they attract each other.
3. Screw in the two arms so they are firmly attached to the base, but can still pivot.

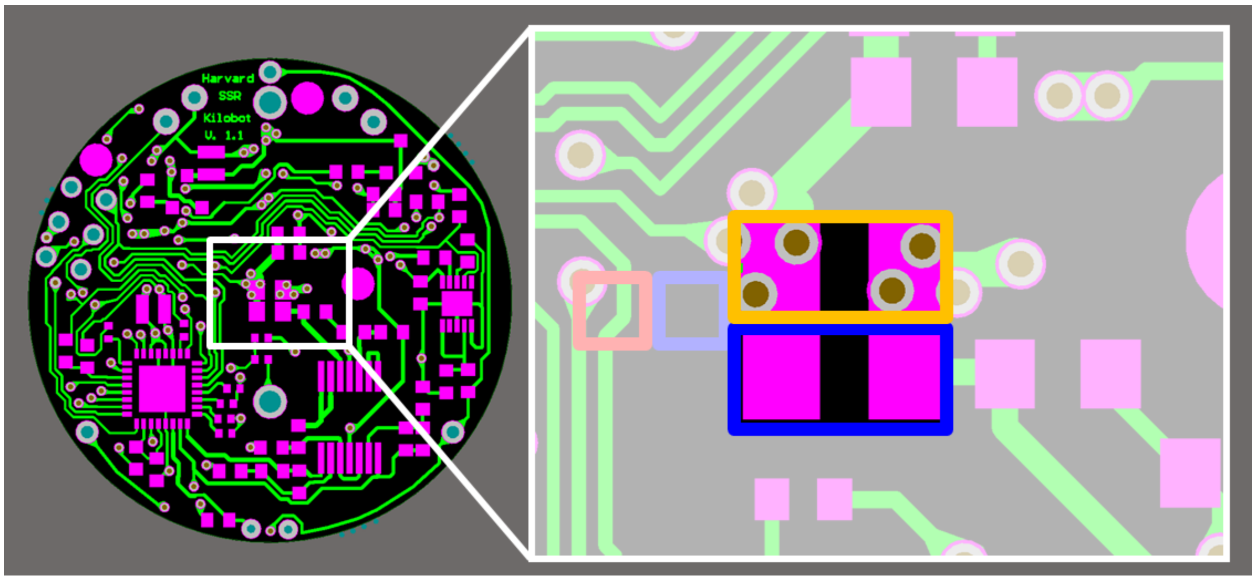


Kilobots Robot Assembly Specification

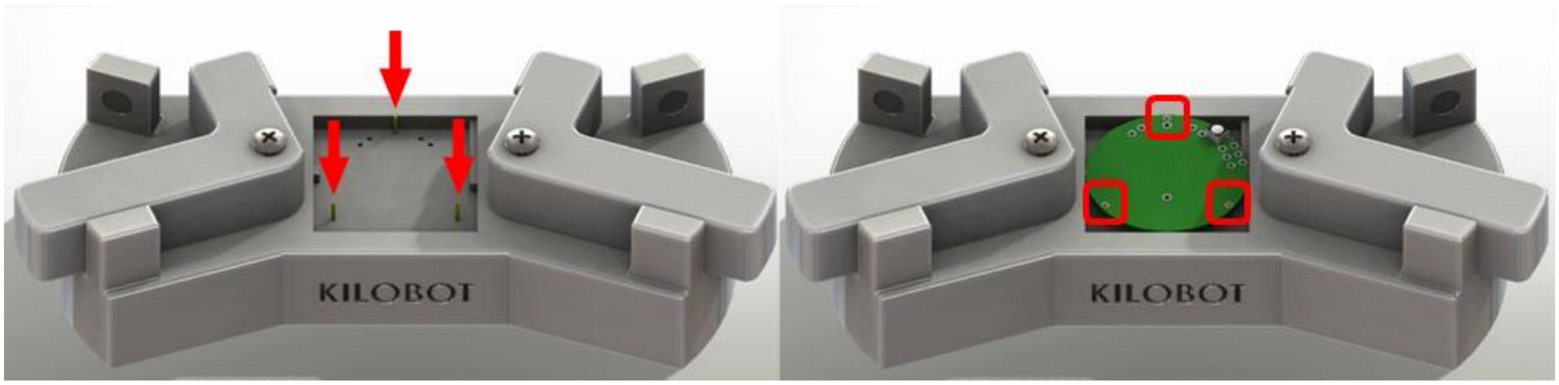
PCB Assembly

This procedure assumes the PCB surface mount components have been pre-assembled by hand or with a pick and place machine. For a bare PCB, components uc1 and charge1 should be soldered in a reflow oven first, and then the remaining surface mount components may be assembled.

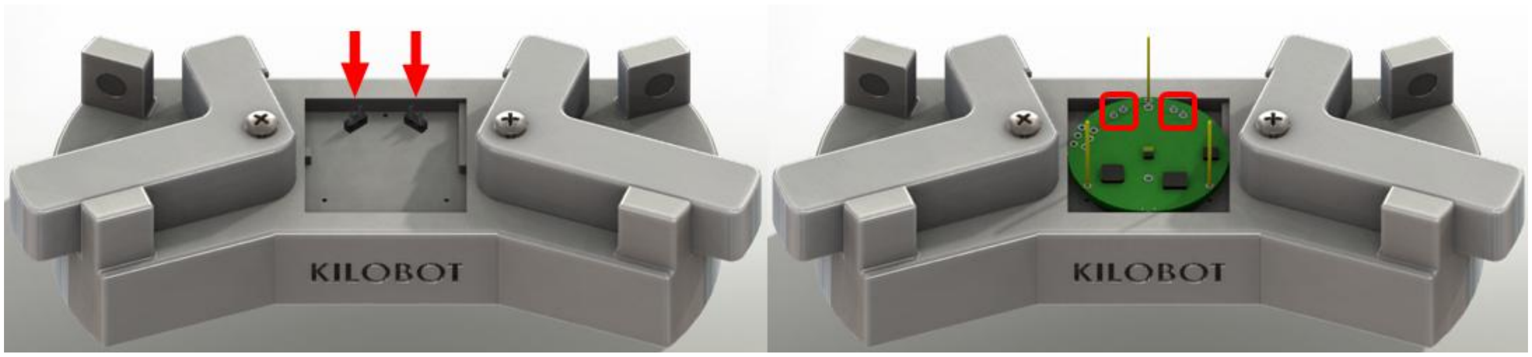
1. After all other surface mount components have been attached, and after the reflow process, solder the infrared transmitter and receiver components to the PCB as shown. Take care to ensure the components are **pressed flat to the board surface**. If not pressed flat, the robot may transmit more light in one direction than others, or be more sensitive to light in one direction, neither of which is desirable. **NOTE: Follow temperature guidelines for these parts when soldering, as not doing so will cause part failures!!**



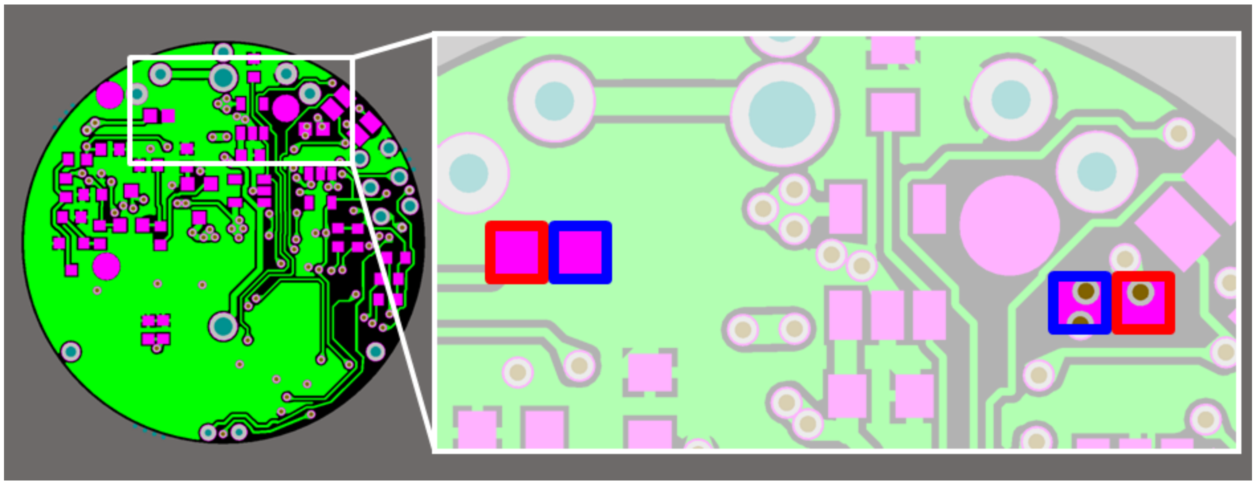
2. Insert three leg pins into the leg pin jig as shown, pushing them down until the touch the table. Place the PCB in the jig as shown and solder the legs in place (the RGB led should be facing up).



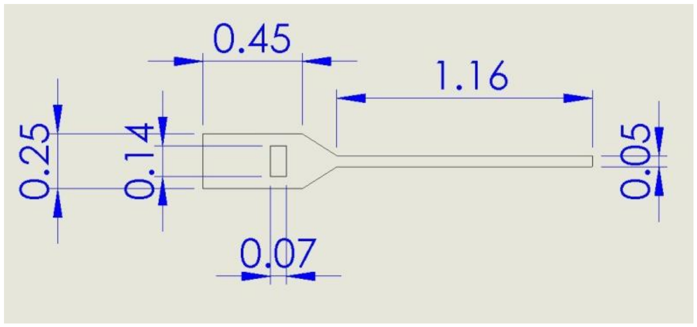
3. Insert two pin headers into the leg pin jig as shown. Place the PCB **upside-down** in the jig and solder the headers in place.



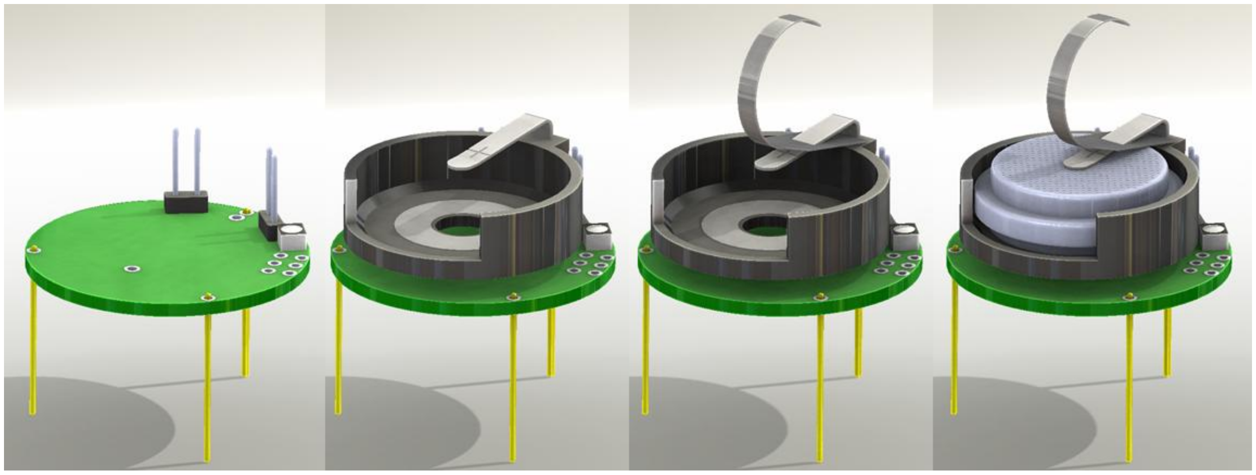
4. Cut the motor leads to be 1.1" (28 mm) in length, strip & tin the tips, and solder onto the PCB top-side as shown. Ensure that the motor leads do not cause any shorts.



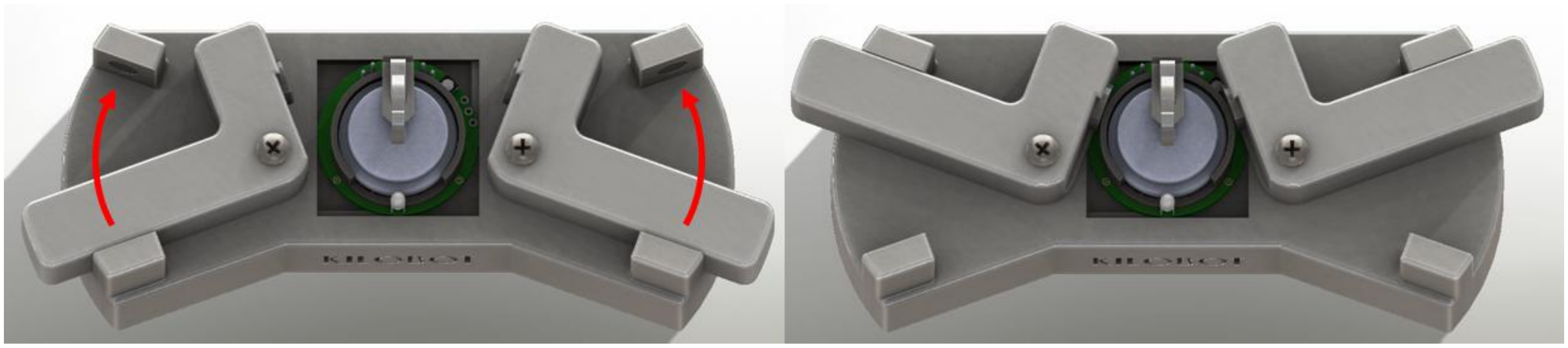
5. The charging tab should be sized according to the drawing below (measurements in inches). We laser cut them out of 4mil spring steel sheets.



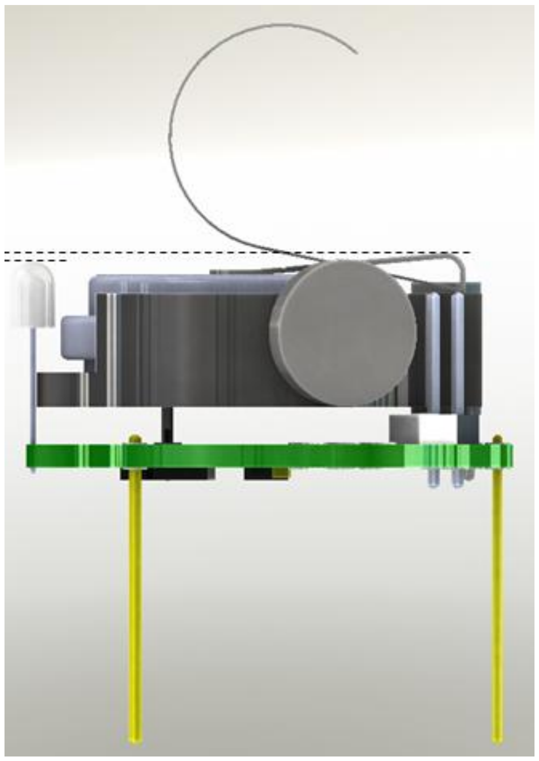
6. Solder the battery clip into the PCB, attach the charging tab, and insert a coin-cell battery **negative side up** as shown.



7. Insert the robot into the motor assembly jig as shown. Attach the top side of each motor onto the round magnet on the jig arms; be sure that they are properly seated in the jig. Carefully apply a small amount of hot glue to the underside of each motor, and rotate the arms into place. Allow 30 seconds for the hot glue to cure.



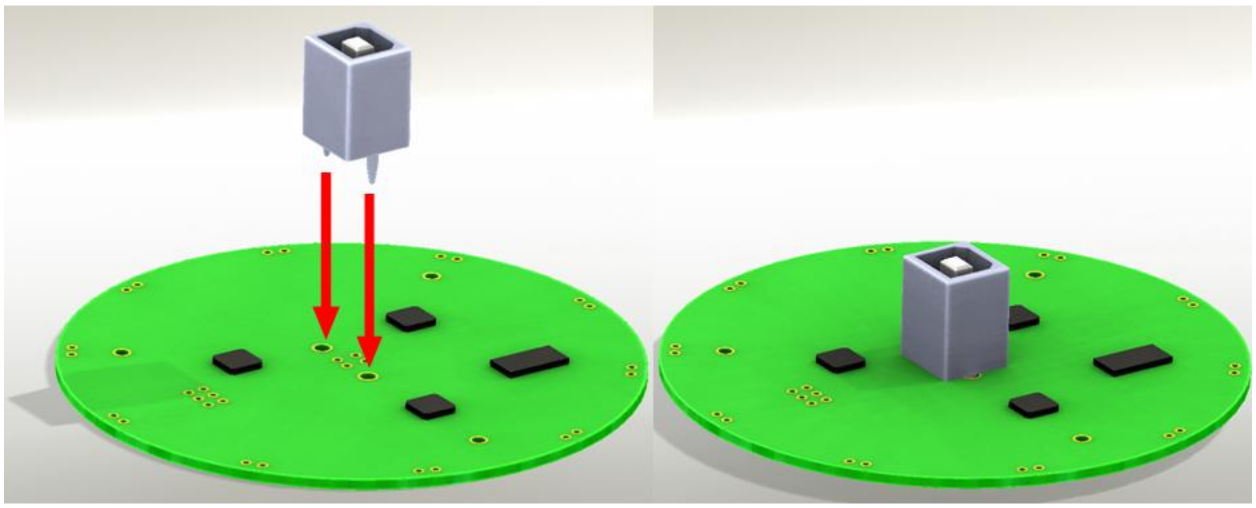
8. Solder the light detector into the PCB top-side so that the top of the lens is just below the height of the battery clip as shown.



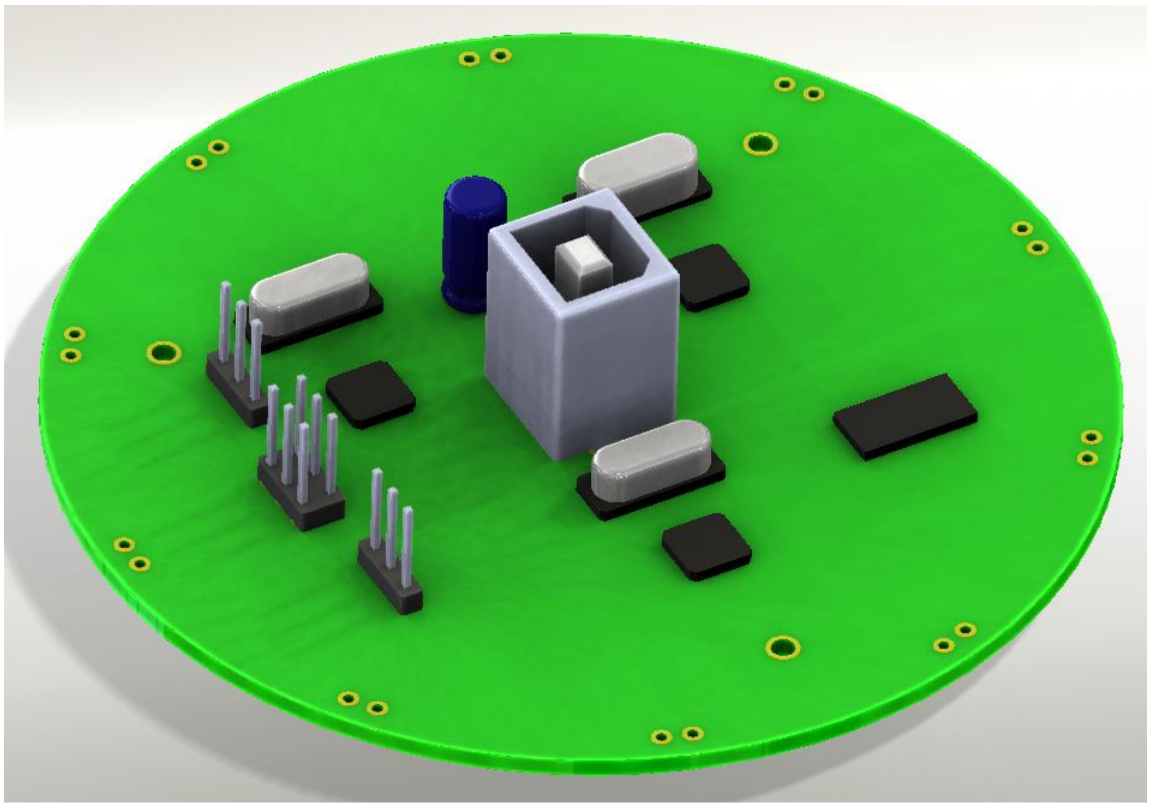
Kilobots OHC Assembly Specification

PCB Assembly

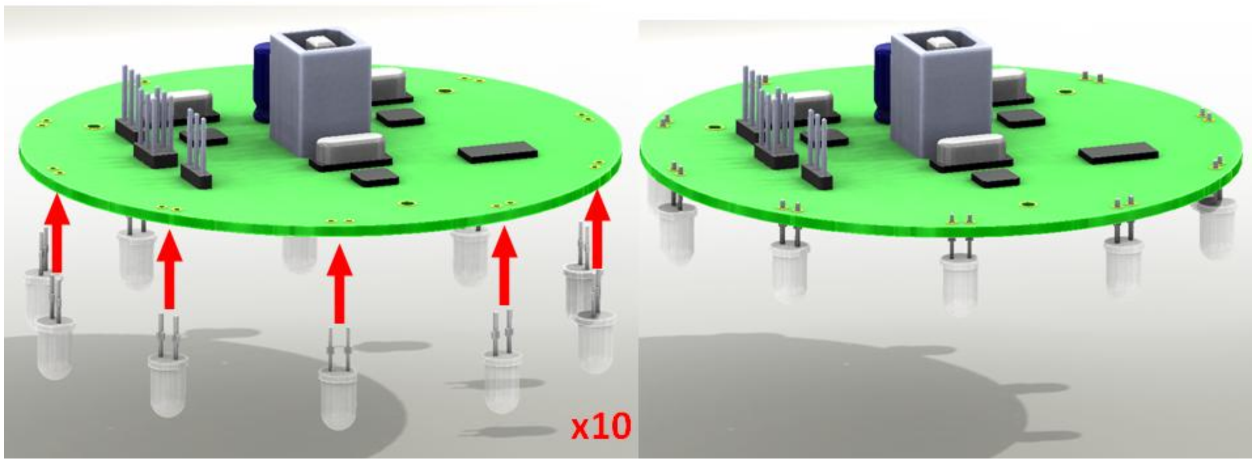
1. Solder the top side surface mount components.
2. Solder USB header to the top side of the board as shown.



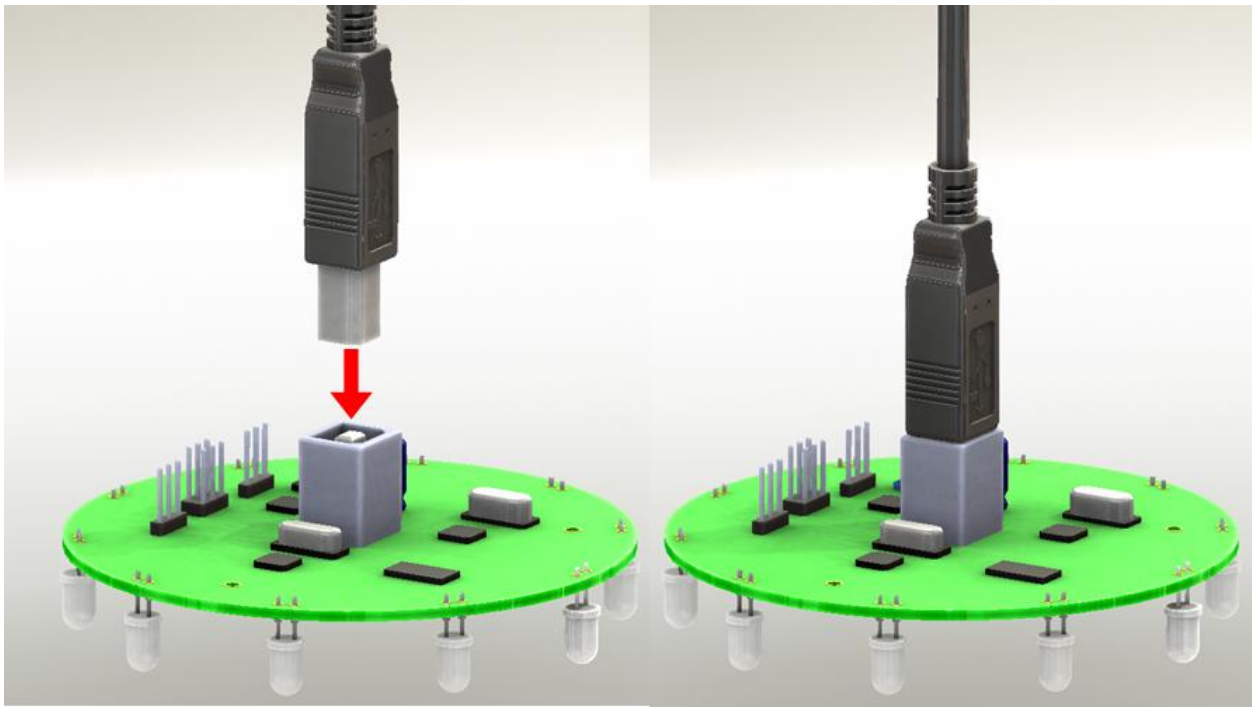
3. Solder the remaining top-side components as shown.



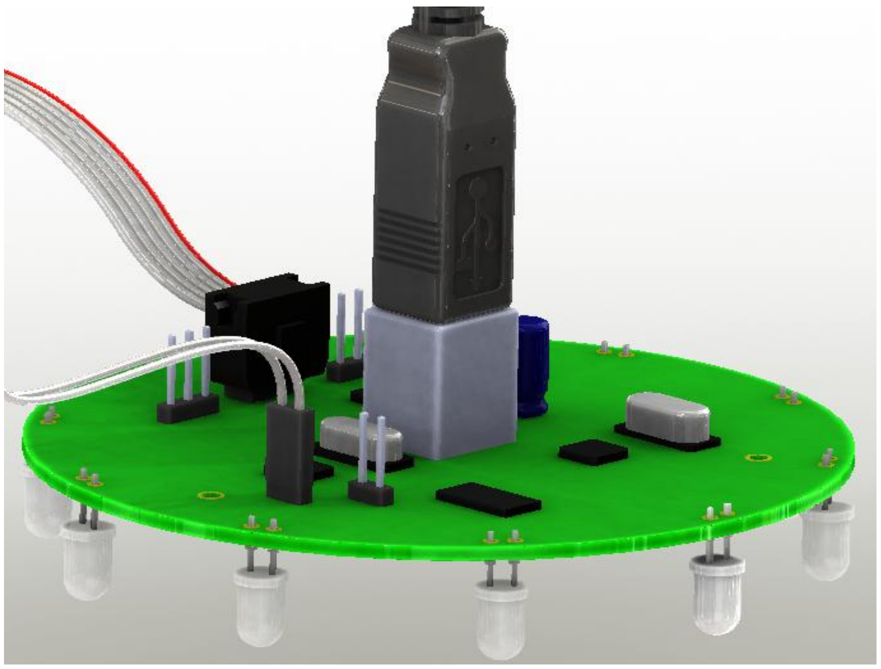
4. Solder the IR LEDs on the bottom side of the board as shown.



5. Insert the USB cable into the header as shown.



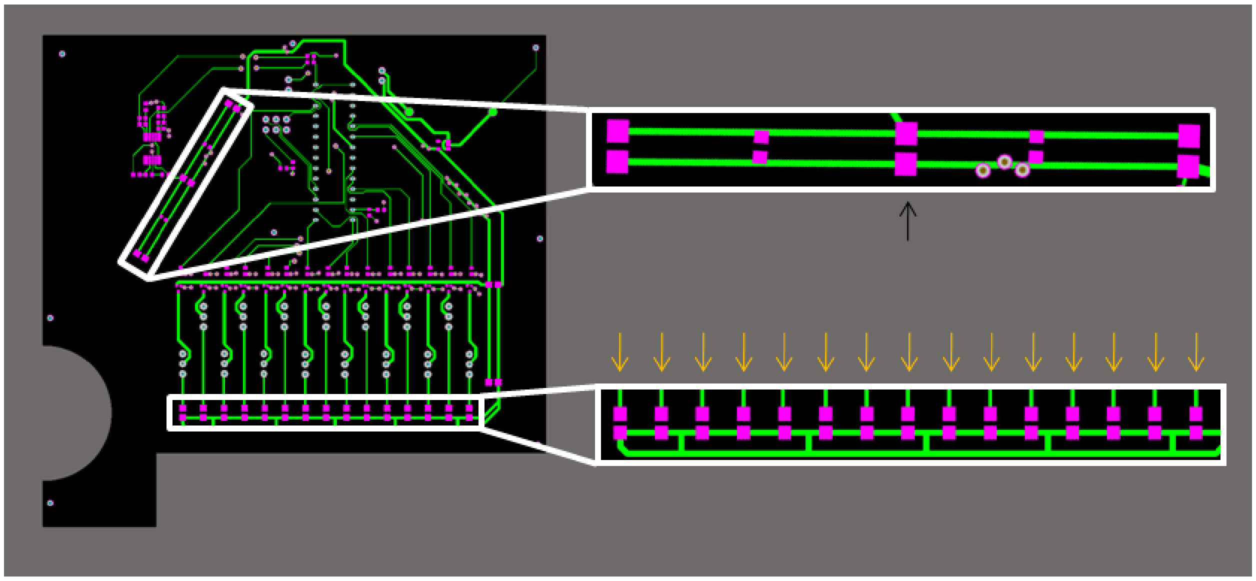
6. Attach kilobot programming cable and serial cable as shown. The programming cable can be used to program a kilobot, and the serial cable can be used to receive serial data from a kilobot, and display it on the computer.



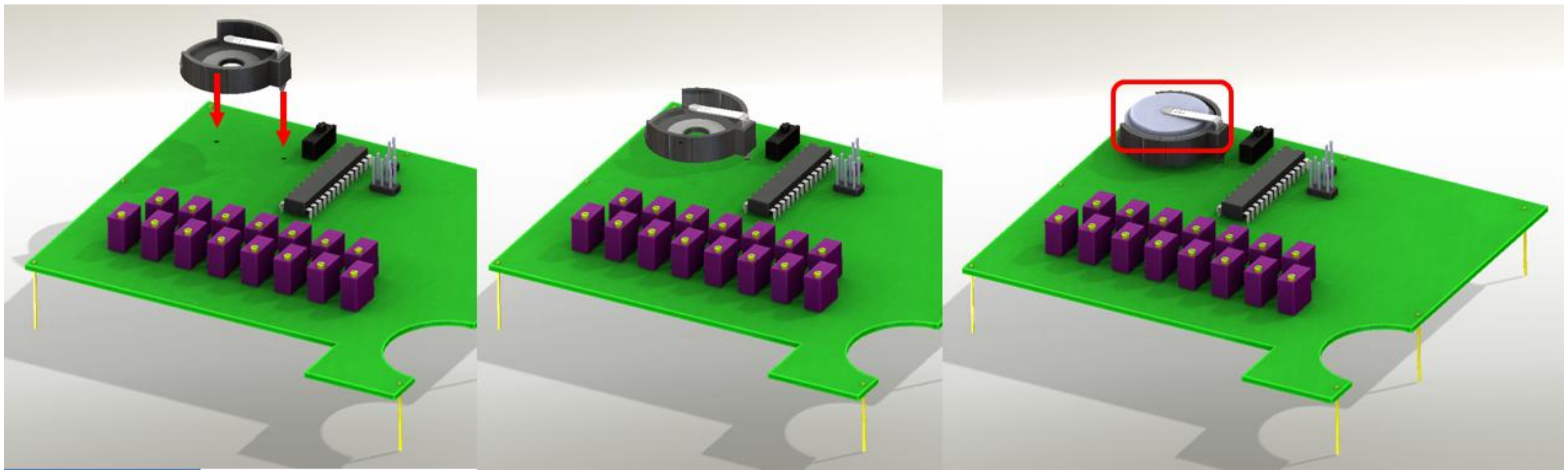
Kilobots Calibration Board Assembly Specification

PCB Assembly

1. Solder the middle infrared receiver and 15 infrared transmitter components to the bottom-side of the PCB. Take care to ensure the components are **pressed flat to the board surface**.

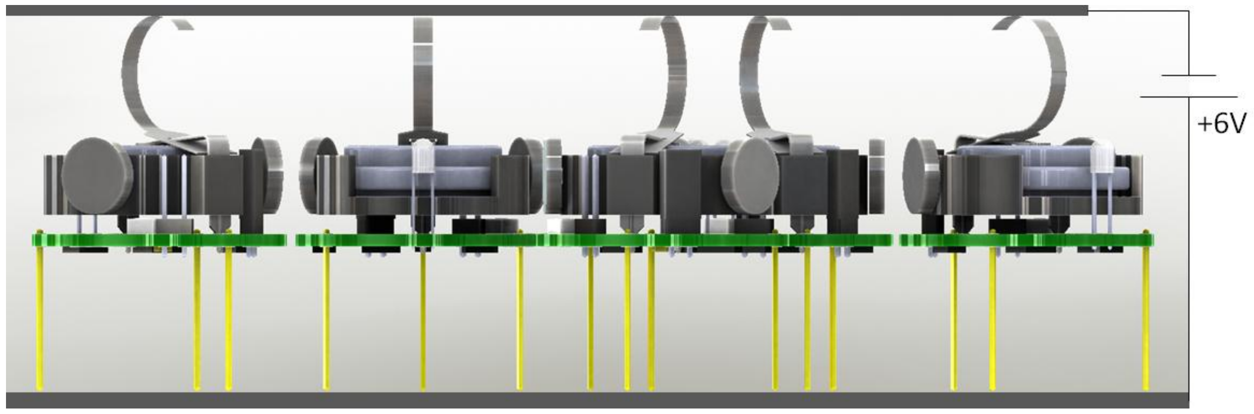


1. Solder the remaining bottom-side surface-mount components.
2. Solder the top-side surface-mount components.
3. Solder the legs to place the calibration board level and at the same height as a Kilobot robot.
4. Solder the through-hole components to the top-side, including the battery clip as shown.



5. Insert coin-cell battery into the battery clip, **positive side up**.

Building a charging station



1. Place a conductive, flat surface (such as a piece of sheet metal) on a table.

2. Apply +6 volts to this surface.

3. Prepare a second conductive plate to place on top of the robot when charging.

4. Attach GND to this second plate.

5. Note: the power supply must be capable of supplying 100 mA/charging robot.

Preparing the Kilobot system for use

After building all the Kilobot system parts, there are some things needed to prepare it for normal operation. These things are 1) loading USB firmware onto the OHC, 2) load kilobot firmware into each robot, 3) tuning the calibration board, 4) calibrate each robot.

Installing OHC firmware

1. Place the unzipped folder KilobotController at c:\
2. Install Atmel's FLIP program. This can be downloaded from Atmel's website. Further in this guide FLIP will be used to transfer the file AVRISP.hex to the overhead controller. The following copyrights apply:

3. To install the USB driver connect the Overhead Controller (OHC) board to the computer. Depending on the Windows version this might trigger the installation of some of the necessary drivers. If the 'Found New Hardware' dialog opens cancel this.

4. Right click the port and update driver. Browse to c:\KilobotController\and select the FTDI32 or FTDI64 folder depending on the Windows version. Install and ignore any request to restart the computer, this is not necessary.

5. The same should be done for the LUFA ISPAVR device if it has not located the driver automatically. Instead of selecting a location, choose 'select driver' and select the AVRISP driver on the list.
AVR studio must be installed for this operation.

6. Open the Flip software (installed in step 2), click on device selection and choose AT90USB162, select the communication mode to be USB. Click load hex and select theAVRISP.hex file. Check Erase, Blank Check, Program and Verify, then Push Run button. Uncheck the Reset button next to Start Application and then click on Start Application. Unplug and reinsert the USB cable. You should now find the AVRISP in the Device Manager under Jungo.

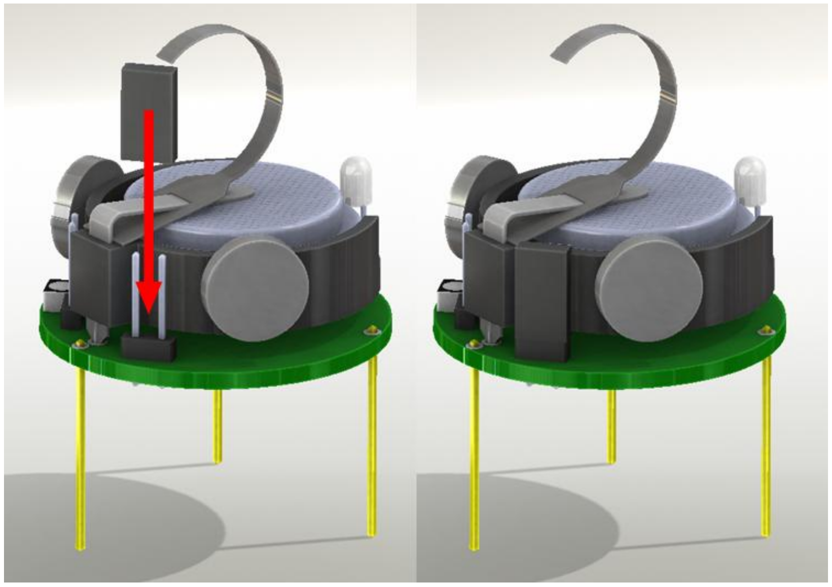
7. Open AVR studio and connect to the OHC. Set the BOOTSZ to start at 3800, enable the BOOTRST fuse, and set the SUT_CKSEL to external 8mhz. Overall fuse settings should be (extended 0xff, high 0xd8, low 0xce).

8. Close AVR studio.

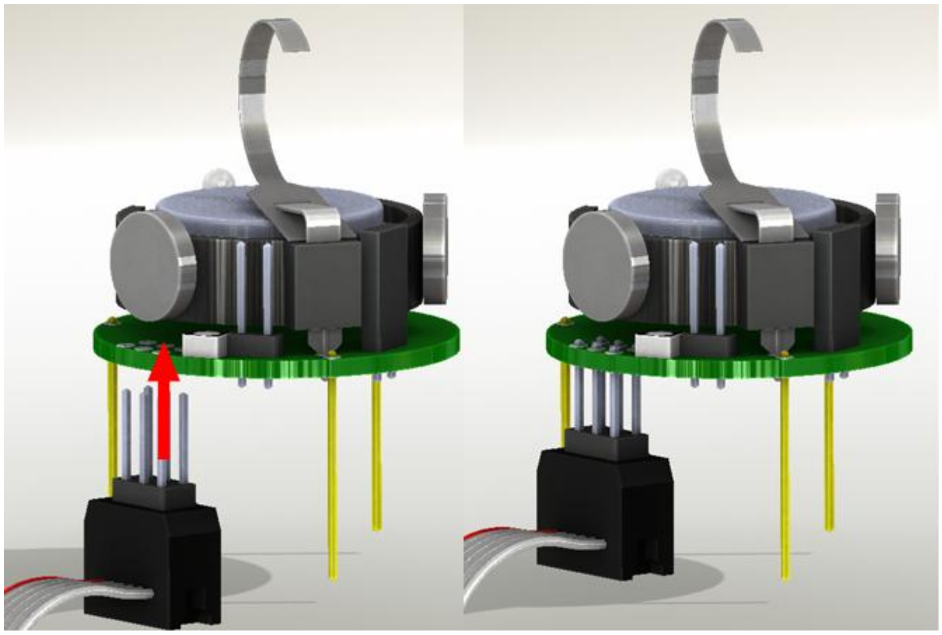
9. Launch the Kilobot Controller application. Select the program (.hex) file that you want to load into the Kilobots. If you want to use an alternative file for the controller instead of the default, click Advance->Select alternative controller file. The default search path for the controller file is c:\KilobotController\controller.hex.
You can now press Program Flash. Check the functionality of the OHC board by toggling the LEDs on the OHC via the GUI.

Kilobot bootloader programming

1. Install WINAVR found here.
2. Install AVR Studio 4 found here (requires registration).
3. In AVR Studio, select Tools->ProgramAVR->Autoconnect
4. In the window that pops up, under the program tab, in the flash category select the input hex file to be *KilobotFirstFirmware.hex*.
5. Turn on the robot by adding the power jumper as shown.



6. Connect the OHC programmer to the robot as shown. Make sure that the programmer pins do not touch the motor on the back side. Gently press the program cable to the side to ensure a good connection.

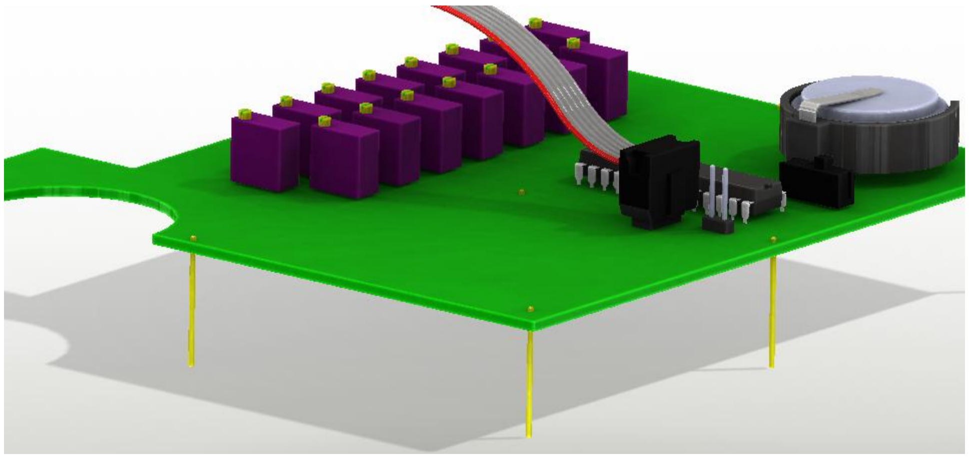


7. set robot fuses. In avr studio AVRISP programming window, go to the fuses tab. Select only the following fuses: spien, EESAVE, BOOTSZ (boot flash size 2048 word, start address =$3800) , SUT_CKSEL (int. RC Osc. 8 MHz; start-up time PWRDWN/RESET: 6 CK/14CK +65 ms). This should result in the following fuse values: extended=0xFF, high=0xD1, low=0xE2
8. In the Autoconnect pop-up window in AVRstudio press program. It should take a few seconds to program the robot, and the robot may vibrate.

Tuning Calibration Board

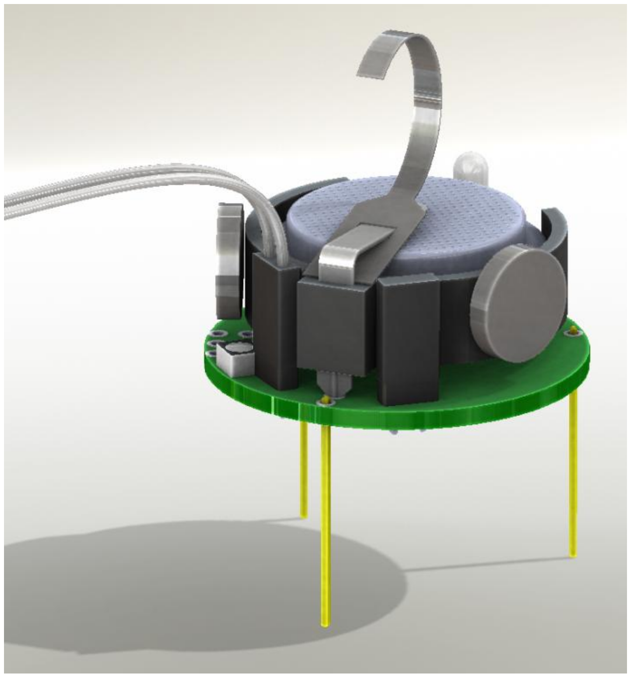
Before calibrating each robot, the calibration board must be tuned. This tuning is done so the calibration board is running at 8mhz, and each of the calibration board transmitters (it has 15 total) is transmitting at the correct intensity. Each led on the calibration board has a Potentiometer that controls the light intensity of the corresponding led.

1. Change calibration board fuse SUT_CKSEL (int. RC Osc. 8 MHz; start-up time PWRDWN/RESET: 6 CK/14CK +65 ms)
2. Load Calibration_time.hex into the calibration board using the OHC programmer as shown.



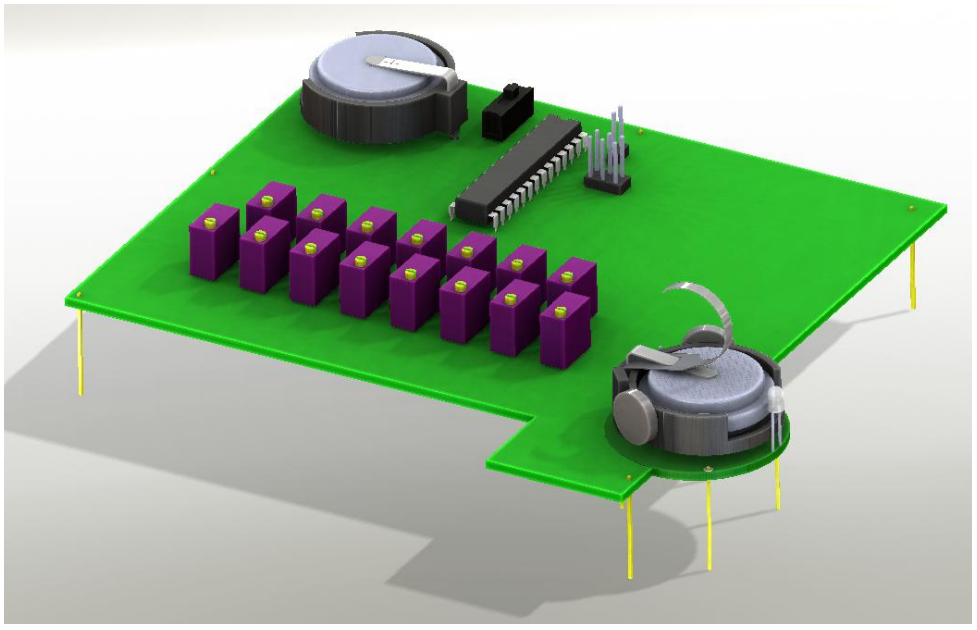
3. Load the controller file calibration_OHC.hex into the OHC (see OHC use section for details how to do this)
4. Connect the serial out cable to the calibration board, so that debug info from the calibration board can be read.
5. Set the computer serial program to receive serial with: Baud rate: 256000, Data: 8 bit, Parity: none, stop: 1, flow control: none

6. Place calibration board under OHC, and power on both. The calibration board will say "complete" when it is done calibrating its timer. This will take about 30 seconds. Be sure to reload the standard OHC code (Controller.hex) into the OHC when done.
7. Take one kilobot with a known good receiver, and load the program Kilobot_cal_rx.hex with the in-circuit programmer.
8. In a second robot with a known good transmitter and load the program Kilobot_cal_tx.hex with the in-circuit programmer.
9. Connect the serial cable from the OHC to the robot running Kilobot_cal_rx .hex as shown. Note that the ground pin for the serial connection is the pin closest to the front leg.



10. Place both robots on the arena, and ensure that both have all three legs touching the arena surface.
11. Place the robots at a distance of 0mm (the robots are touching) from each other, measure the average value from the receiving robot, which is sent via the serial output.
12. Measure this average value for every 5mm distance from 0 to 70.

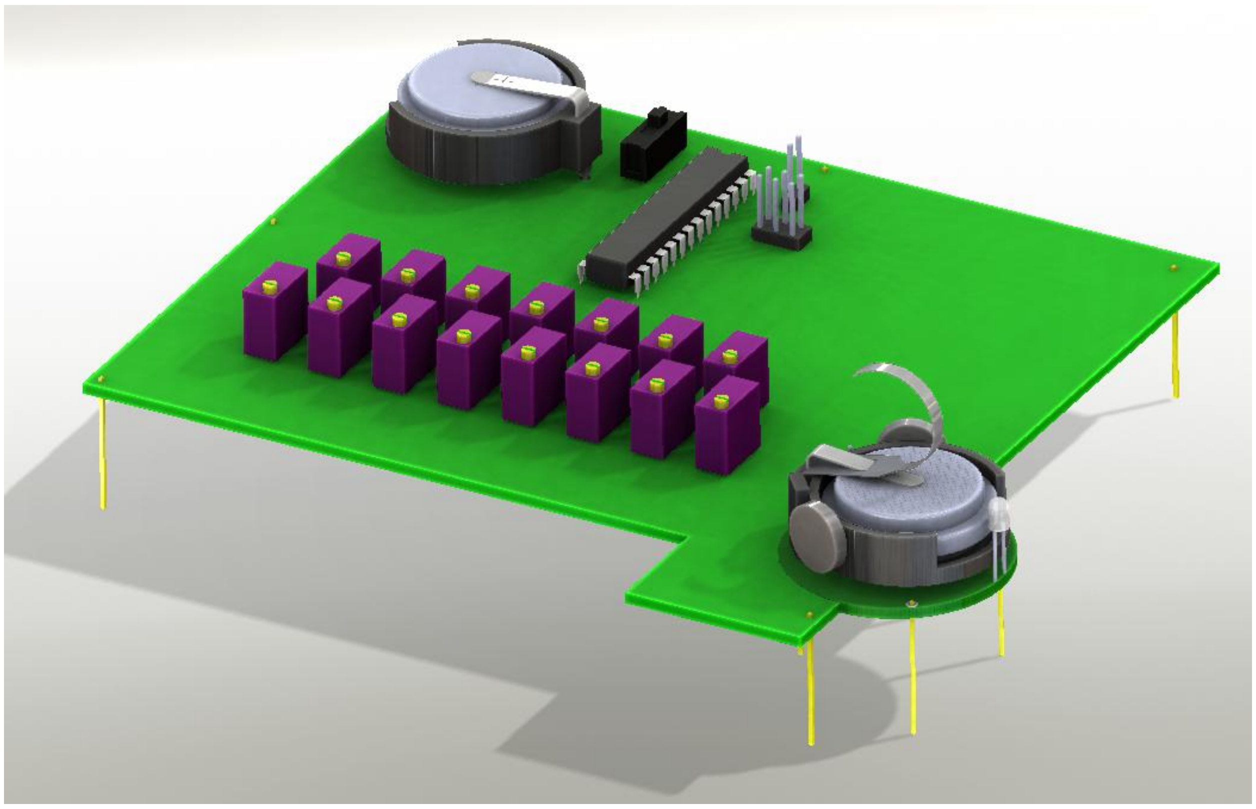
13. Program the robot with Calibration_RX.hex with a new program Kilobot_calibration_board_calibrate.hex, and connect it to the serial out cable.
14. Program the calibration board with Calibrationboard_txonly.hex.
15. Place the robot with Kilobot_calibration_board_calibrate.hex as shown below.
16. Attach serial cable to this kilobot.
17. Adjust each potentiometer until it the values from serial out match those recorded in steps 7 and 8.



Kilobots Calibration Procedure

IR Calibration

1. Upload *KilobotCalibration.hex* program to the robot.
2. Load Calibration_board.hex to the calibration board.
3. Place the robot in the semi-circular cutout of a calibration board.

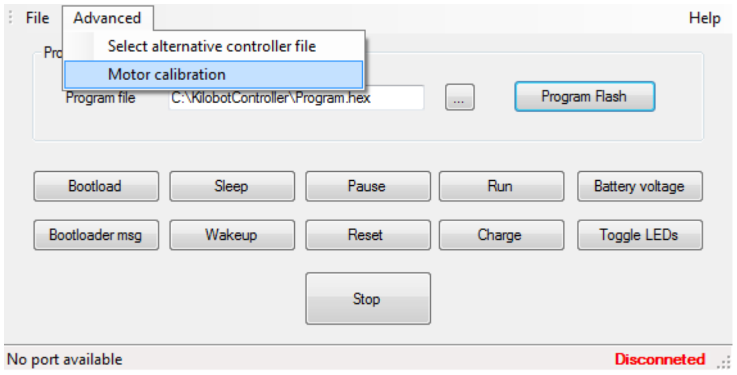


4. The robot will automatically calibrate, taking about 30 seconds. The LED will blink during the calibration. If calibration is successful, the LED will be a steady green for 10 seconds and turn off (robot in SLEEP mode).
5. If the calibration is not successful, the LED will be a steady red. Try restarting the robot and the calibration board.

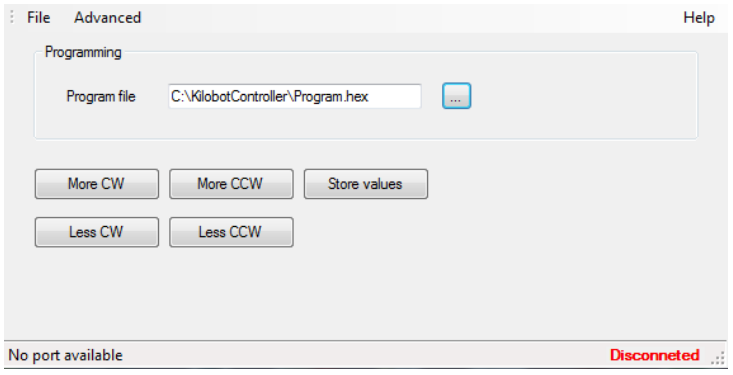
Motor Calibration

1. Upload motor calibration receiver program Kilobot_motor_cal.hex to the robot.
2. Upload motor calibration transmitter program OHC_motor_cal.hex to the OHC.

3. In the OHC software running on the computer, select the motor calibration menu as shown.



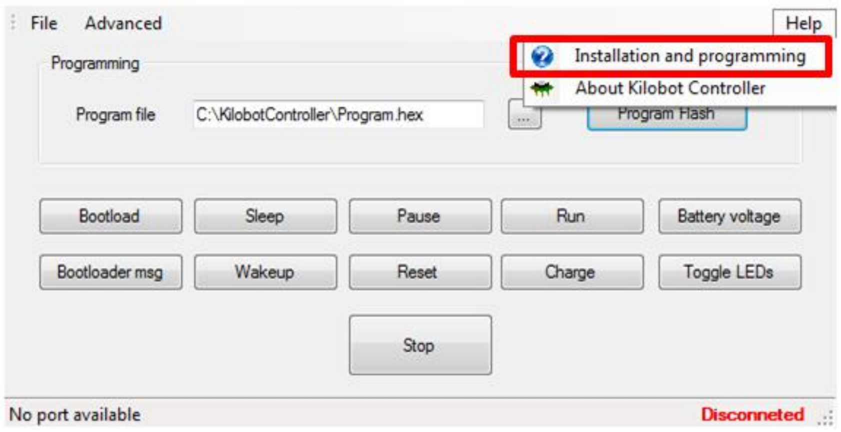
4. Place the kilobot under the OHC, and click the button "more cw" or "less cw" to adjust the calibration of the clock wise rotation for the robot.
5. When clock wise rotation is good, click "more CCW" once. Now you can use the "more CCW" and the "less CCW" buttons to calibrate the robots CCW rotation.
6. When CCW rotation is good, click the "more cw" button once. now you can use the "more CCW", "less CCW", "more CW", "less CW" buttons to adjust the robots straight calibration value.
7. When the robots straight movement is good, click "store values". The robot will store calibration values in eeprom, and then go to sleep. The motors are now calibrated.
8. When done, be sure to load Controller.hex back into the OHC.



Kilobots Software Installation Guide

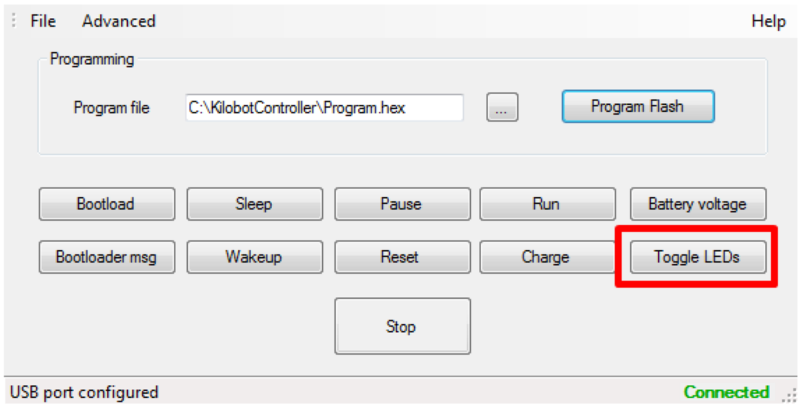
Overhead Controller (OHC) Software Installation

1. Place *KilobotController* folder in C: (so the location is C:\KilobotController)
2. Open *KilobotController.exe*
3. In the Help menu of *KilobotController.exe*, select "Installation and programming" and follow the instructions to install the software and drivers.



Overhead Controller (OHC) Hardware Installation

1. Connect both USB cables to the PC.
2. Toggle the OHC power switch to turn it on.
3. Run the *KilobotController.exe* program, and click the "Toggle LEDs" button to test that a connection is made. This should blink the blue LEDs on the controller.



General Robot use

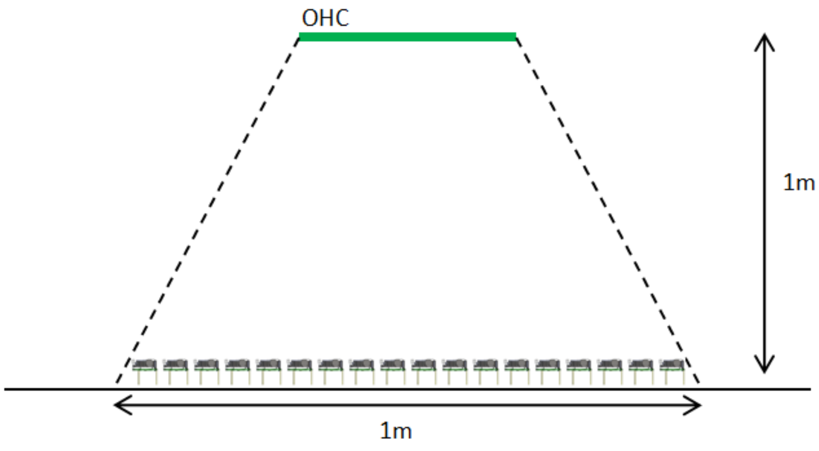
This section assumes the Kilobot system has been built and prepared for general use. It will explain how the setup and use of the OHC, how to charge the robots, and how to write programs for the robots.

OHC Use

System Setup

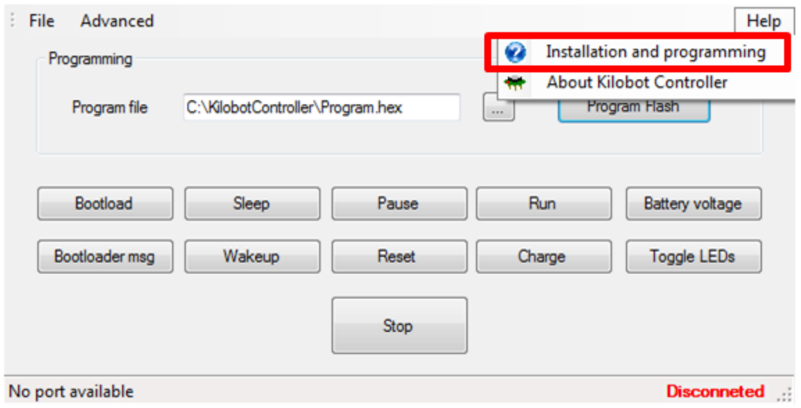
Kilobots should be operated on a smooth, flat surface to ensure proper robot mobility. To aid communication, the surface should be glossy or reflective. A mirror or dry-erase whiteboard oriented horizontally is recommended.

The overhead controller should be hung above the Kilobots at a distance of about one meter. The robots beneath the OHC in a about a one meter diameter region will be able to receive communication from the OHC as shown.



Overhead Controller (OHC) Software Installation

4. Place *KilobotController* folder in C: (so the location is C:\KilobotController)
5. Open *KilobotController.exe*
6. In the Help menu of *KilobotController.exe*, select "Installation and programming" and follow the instructions to install the software and drivers.



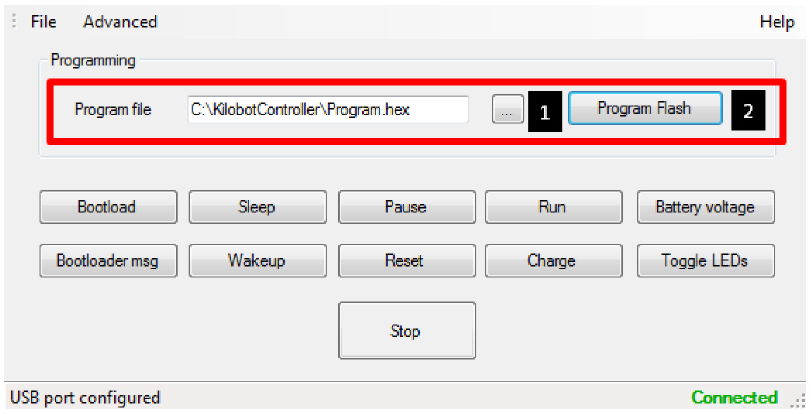
Program file C:\KilobotController\Program.hex

No port available

Disconneted

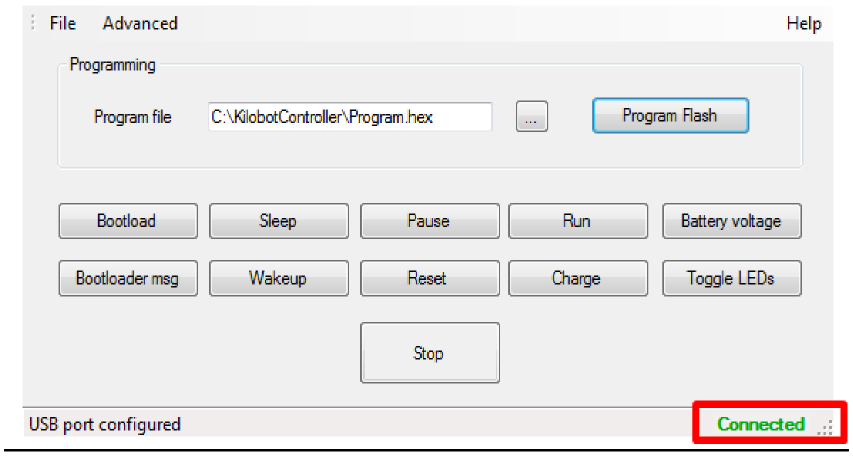
Overhead Controller (OHC) Interface Overview

Programming Panel



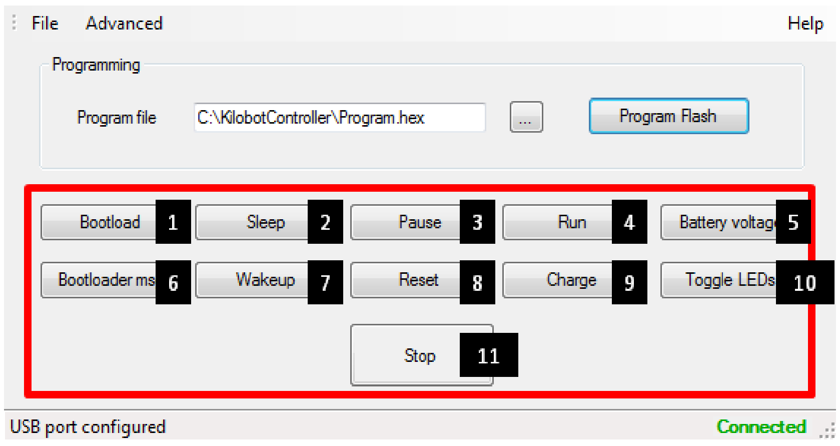
1. Program file: browse for the compiled kilobot user program [.hex] file to be sent to the Kilobots
2. Program Flash: send the program and controller files to the OHC.

Connection Status



1. Indicates weather the OHC is connected to the pc or not.

Button Panel



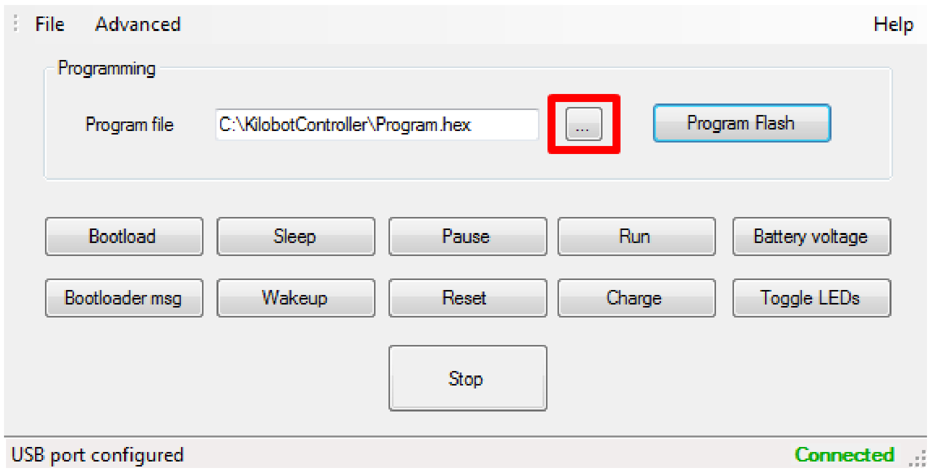
1. Bootload: transmit the kilobot user program from the OHC to the Kilobots, which takes about 30 seconds. Ensure all robots are in pause mode before bootloading.
2. Sleep: put Kilobots in low-power SLEEP mode. They will wake up breifly every 60 seconds to check if a wakeup message is being sent. The robots conserve energy in this mode and can stay in sleep for months without recharging.
3. Pause: put Kilobots in idle PAUSE mode, and is also used to prepares robots to be repogrammed.
4. Run: initiates Kilobots user program.
5. Battery Voltage: instructs Kilobots to indicate their battery voltage through the LED. where green indicates a battery voltage over 4V, blue indicates a voltage between 3.75 and 4, yellow indicates a voltage between 3.5 and 3.75, and red indicates a voltage under 3.5 volts
6. Bootloader msg: sends a message for all robots to exit their program and jump to the bootloader
7. Wakeup: take Kilobots out of low-power SLEEP mode. This message will transmitt untill stop is pressed. It will take a minute or two for all the robots to wake up
8. Reset: restarts the Kilobot program
9. Charge: instructs Kilobots to enter CHARGE mode for battery recharging. The robots will stop the user program and blink red if they are activly charging, if not, the RGB led will turn off.
10. Toggle LEDs: flashes the LEDs on the OHC.
11. Stop: stops whatever the OHC is doing. This can be used to stop functions that are continuous such as wakeup or bootloading.

Programming a kilobot group with a new program

1. In AVRstudio open the “Kilobot” project.
2. In *UserProgramSkeleton.c* locate the following section

```
/////////////////////////////////////////////////////////////////////////////////////
//user program code goes here.  this code needs to exit in a reasonable amount of time
//so the special message controller can also run
/////////////////////////////////////////////////////////////////////////////////////
```

3. Immediately after this is where the user code is written. Some examples of the API are given in comments. Note that the user code should not contain long blocking functions, otherwise the robot may not respond to the overhead controller properly.
4. Once program is written, in AVRstudio select Build->build.
5. Run *KilobotController.exe*, click the button to browse for a program file as shown and select the program [.hex] file you just built (located at *kilobot/default/UserProgramSkeleton.hex*).



C:\KilobotController\Program.hex

USB port configured

Connected

6. Press the “Program Flash” button to program the OHC (a black window will pop up).
7. Place *ALL* Kilobot robot in PAUSE mode (LED flashing yellow) underneath the OHC, and press the “Bootload” button in *KilobotController.exe*. The Kilobot will first quickly flash red→green→blue, to indicate it is now in the PROGRAMMING mode. Within 5 seconds, the LED will begin to flash blue and continue flashing while it is receiving the new program. Once the LED has stopped flashing blue and switche to the flashing yellow of pause mode(approximately 30 seconds), press the “Stop” button in *KilobotController.exe*.
8. To run the program on the robot, press the “Run” button in *KilobotController.exe*. The new program should execute.
9. Press the “Pause” button in *KilobotController.exe* to stop the program. (Note: the robot needs to be in pause mode before programming can begin).

Kilobots API

//set_motor(cw_motor,ccw_motor)
//seet motor speed PWM values for motors between 0 (off) and 255 (full on)
//Example:
set_motor(100,100);

note that there are 4 calibration values used with the motors

```
cw_in_place - value for cw motor to turn the robot cw in place (note: ccw motor should be off)
ccw_in_place - value for ccw motor to turn the robot ccw in place (note: cw motor should be off)
cw_in_straight - value for the cw motor to move the robot in the forward direction
ccw_in_straight - value for the ccw motor to move the robot in the forward direction
```

//_delay_ms(x);
//busy wait for x milliseconds, interrupts can still trigger
//max value for milliseconds = 4000
//Example:
_delay_ms(250);

//set_color(r,g,b);
//set LED color, values can be from 0(off)-3(brightest)
//Example:
set_color(1,1,1);

//kprinti(int);
//print integer over serial port - be careful can effect timing!
//Example:
kprinti(12);

//kprints(string);
//print string up to 10 characters - be careful, can effect timing!
//Example:
kprints("HelloWorld");

//message_out(tx0,tx1);
//set message values to be sent over IR every .2 seconds, 2 bytes tx0,tx1
//Example:
message_out(0x01,0xFF);

//get_message();
//take oldest message off of rx buffer
//message is only new if message_rx[5]==1 !
//if so, message is in message_rx[0],message_rx[1]
//distance to transmitting robot (in mm) is in message_rx[3]
//Example:
get_message();

```
if(message_rx[5]==1)
{
  data=message_rx[0];
  distance=message_rx[3];
}


//enable_tx
//to turn off the transmitter, set enable_tx = 0
//to turn on the transmitter, set enable_tx = 1
//Example:
if(Want_To_Transmit==TRUE)
        enable_tx = 1;
else
        enable_tx = 0;


//measure_charge_status();
//measure if battery is charging, returns 0 if no, 1 if yes
//Example:
if(measure_charge_status())
        charging = TRUE;
else
        charging = FALSE;


//measure_voltage();
//measure battery voltage, returns voltage in .01 volt units
//for example if 394 is returned, then the voltage is 3.94 volts
//Example:
current_voltage = measure_voltage();


//get_ambient_light();
//returns the value of ambient light
//note: will return -1 if there is an incoming message (which also uses a/d)
//note: turns off interrupts for a short while to sample a/d
//Example:
local_brightness = get_ambient_light();


//numerous other functions and AVR-C programming examples can be found on the web.
```

Example code

transmit to neighbors, and blink led when message is received

```
message_out(255,255);//set message to be sent over IR
enable_tx=1;//enable transmission of message every .2 seconds

//check for message
get_message();
if(message_rx[5]==1)//new message has been received
{
        set_color(1,1,1);//turn RGB LED white
        kprinti(message_rx[0]);//send first byte of received message over serial debug cable
        kprinti(message_rx[1]);//send second byte of received message over serial debug cable
        kprinti(message_rx[3]);//send measured distance from transmitting robot over serial debug cable
        kprints("        ");
        _delay_ms(10);//wait 10 ms
        set_color(0,0,0);//turn RGB LED off

}
```

Non-blocking timed movement

```
static int mode=0;//mode represents state
if(clock<40000)
{
        //clockwise rotation
        if(mode!=0)
        {
                set_motor(0,0xa0);//spin up cw motor to overcome friction
                _delay_ms(30);
                set_motor(0,cw_in_place);//set cw motor to calibrated value for good cw rotation
                mode=0;
        }


}
else if(clock<80000)
{
        if(mode!=1)
        {
                //counter clockwise rotation
                set_motor(0xa0,0);//spin up ccw motor to overcome friction
                _delay_ms(30);
                set_motor(ccw_in_place,0);//set ccw motor to calibrated value for good cw rotation
                mode=1;
        }


}
else if(clock<120000)
{
        if(mode!=2)
        {
                //move forward
                set_motor(0xa0,0xa0);//spin up both motors to overcome friction
                _delay_ms(30);
                set_motor(ccw_in_straight,cw_in_straight);//set both motor to calibrated value for good forward movement
```

```
                    mode=2;
          }
}
else if(clock<160000)
{
          if(mode!=3)
          {
                    set_motor(0x00,0x00);//turn off motors
                    mode=3;
          }
}
else if(clock<200000)
{
          mode=5;
          clock=0;//reset clock to start cycle over agian
}
```

Random move in see neighbors, stop else

```
static int one_time=0;
if(one_time==0)//if this is the start of the program, only run this if statement once
{
          for(int i=0;i<30;i++)
                    randseed+=measure_voltage( );//generate some random sensor data
          one_time=1;//indicate that the first time if statement has now run
          srand(randseed);//seed random variable with some sensor data
}
static int neighbor=0;//1 if i've seen any neighbors, 0 if not
static int movement=0;//1 if moving, 0 if stopped
static int start_move=0;//1 if the movement has changed, 0 if not

//check to see if i have received any new messages, if so, then i have seen neighbors
get_message();
if(message_rx[5]==1)
          neighbor=1;

//if i am currently not moving, and want to start moving
if(start_move==1)
{
          if(movement==0)
          {
                    set_motor(0xa0,0xa0);//spin up motors
                    _delay_ms(30);
                    set_motor(ccw_in_straight,cw_in_straight);//set to move straight
          }
          else if(movement==1)
          {
                    set_motor(0xa0,0);//spin up motor
                    _delay_ms(30);
                    set_motor(ccw_in_place,0);//set to move ccw
          }
          else
          {
                    set_motor(0,0xa0);//spin up motor
                    _delay_ms(30);
                    set_motor(0,cw_in_place);//set to move cw
          }
          start_move=2;//mark that i am currently moving

}
else if(start_move==0)//i do not want to move
{
          set_motor(0,0); //stop motors
}
```

```
//every 2000 counts of clock, check to see if i've seen any neighbors
//if i have, there is a 20% chance i will change my movement (such as straight to ccw)
if(clock>2000)
{
        if((rand()%100)<20)//update robot movement 20% of the time
        {
                int rand_d=rand()%100;
                if(rand_d<25)//if updating robot movement 25% of the time move ccw
                {
                        if(movement!=1)
                        {
                                movement=1;
                                if(start_move==2)
                                start_move=1;
                        }
                }
                else if(rand_d<50)//if updating robot movement 25% of the time move cw
                {
                        if(movement!=2)
                        {
                                movement=2;
                                if(start_move==2)
                                        start_move=1;

                        }
                }
                else//if updating robot movement 50% of the time move straight
                {
                        if(movement!=0)
                        {
                                movement=0;
                                if(start_move==2)
                                        start_move=1;

                        }
                }
        }
        //if i've received a message from a neighbor in the last 2000 counts of clock
        //and i am not moving, start moving, and turn my rgb led red
        if(neighbor==1)
        {
                set_color(1,0,0);
                if(start_move==0)
                        start_move=1;
        }
        else
        {
                //if i have not received a message from a neighbor in the last 2000 counts of clock
                //stop moving, and turn my RGB led off
                set_color(0,0,0);
                start_move=0;
        }

        clock=0;//reset clock
        neighbor=0;//reset neighbor count

}
```