ASEE's 123rd Annual • Conference & Exposition • New Orleans, LA • June 26-29, 2016

New Orleans JAZZED about Engineering Education

Paper ID #15441

MAKER: A Kilobot Swarm

Mr. Nathan Tyler Thomas, Western Carolina University
Dr. Yanjun Yan, Western Carolina University

Yanjun Yan received her B.S. and M.S. degrees in Electrical Engineering from Harbin Institute of Technology (China), and the M.S. degree in Applied Statistics and the Ph.D. degree in Electrical Engineering from Syracuse University. She is an assistant professor in engineering and technology at Western Carolina University. Her research interests are statistical signal processing, diagnostics, and particle swarm optimization.

Dr. Hugh Jack, Western Carolina University

Dr. Jack is the Cass Ballenger Distinguished Professor of Engineering and Department Head of the School of Engineering and Technology within Western Carolina University. His interests include robotics, automation, and product design.

MAKER: A Kilobot Swarm

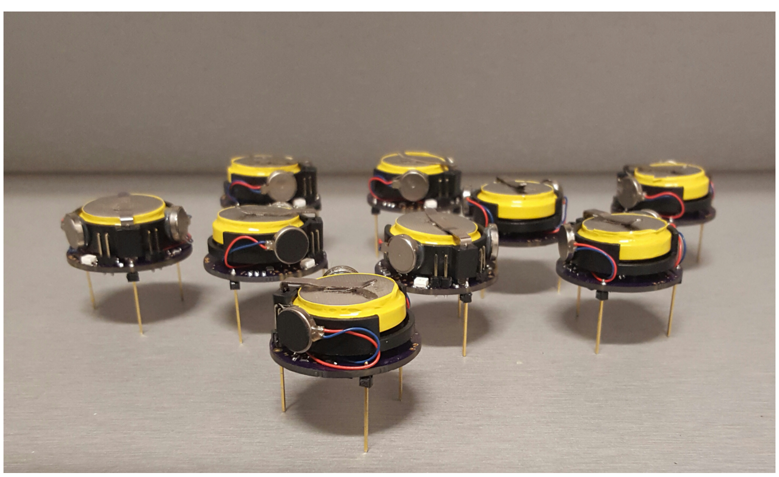
Abstract

A Kilobot is a small, inexpensive robot designed by the Self-Organizing Systems Research Group at Harvard University. These robots have features that enable researchers to test collective algorithms on hundreds of robots (called a swarm) without the logistical problems that are faced when dealing with a large number of units[1]. This paper presents a tested procedure on how to construct a Kilobot Swarm. The procedure was created from a combination of the original Kilobot documents made available by Harvard University[2], other materials later released by Harvard[3], as well as some additional insight and modifications that result in a single document with all of the relevant information to get a Kilobot Swarm operational.

Introduction

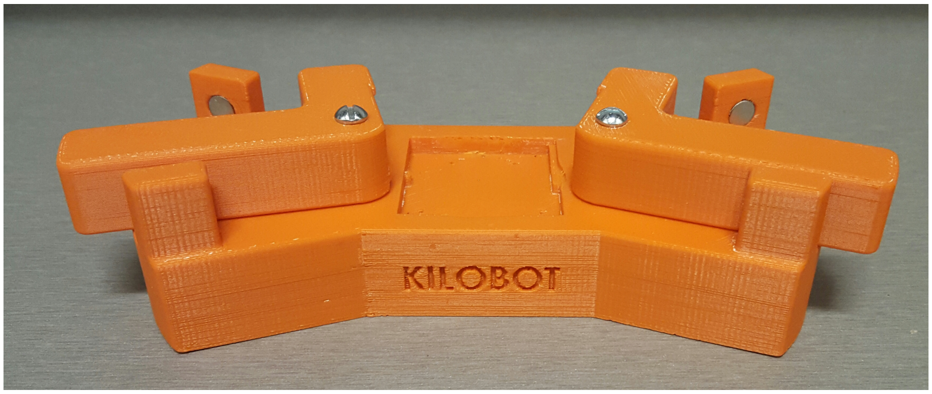
The components of a Kilobot Swarm include: multiple Kilobot agents, an assembly jig, the arena, and the overhead infrared programmer/controller (OHC). A Kilobot agent is a small robot about 1.25" in diameter with three legs that make it stand about 1.5" tall. The legs work in tandem with two vibration motors mounted to the sides. These vibration motors cause the legs to walk independently giving the unit the ability to move in a differential manner[1].

Figure 1: Kilobot Agents



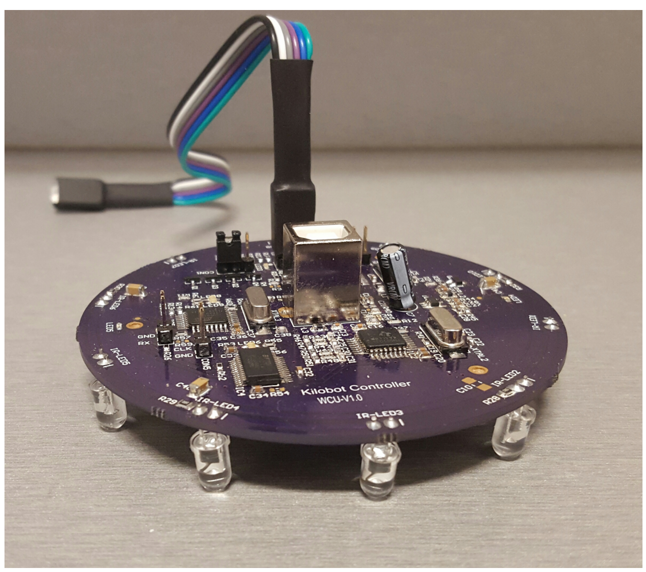
The assembly jig is a 3D printed component that assist in the Kilobot construction, making the mounting of the legs and motors quick and consistent.

Figure 2: Kilobot Jig



The arena can consist of any smooth reflective surface but, as recommended, a dry erase surface should be used. The overhead infrared programmer/controller (OHC) is a circular printed circuit board (PCB) about 3” in diameter with a USB connection on the top center and infrared LEDs mounted on the bottom perimeter of the PCB. The OHC is designed to program or control multiple Kilobots at one time.

Figure 3: OHC Model



Required skills for this project:

- advanced SMD soldering and reflow techniques.
- basic C or C++ programming.

- basic knowledge of Atmel products.
- basic knowledge of 3D printing and modeling.

Required equipment for this project:

- Windows PC.
- soldering iron suited for small components.
- reflow oven.
- 3D Printer.

Design Materials

The documents and software used in this procedure have been compiled from two sources: the original Kilobot documents made available by Harvard University at `http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html` and updates later released at `https://www.kilobotics.com/`.

The first source of information is the *Kilobot_ documents* folder that contains most of the relevant information needed to get a batch of Kilobots operational. The download will be labeled *Kilobot_ documents* and will contain 17 items of different types. All .hex and .c files (there should be 13 total) should be placed in a new folder labeled *System Software*, then drag the *KilobotController* folder in to this new folder as well. This folder now contains all of the software needed for the system. Some of these files may not be used, but it is recommended to keep them all for later use. Now, in the main directory of *Kilobot_ documents* there should be two folders (*DESIGN FILES* and *System Software*), a PDF labeled *Kilobot Guide* and a .txt file labeled *Readme*. *Readme* contains a link for license information explaining acceptable uses of the material. The *DESIGN FILES* folder contains 6 items; three folders and three .SLDPRT files. The three folders contain the hardware design information (schematics, PCB layout, and CAM files) for the kilobots, the OHC, and the calibration unit (not used in this process). The three .SLDPRT files labeled *Assembly_ Jig1*, Assembly_ Jig2, and Assembly_ Jig3 are 3D part files from a 3D modeling software known as SolidWorks. These three files can be converted into .stl files and printed to make the jig that will assist in placing the motors and legs on each kilobot. Lastly, *Kilobot Guide* is the main reference document for the assembly and operation of the system. All relevant information from this document has been revised and presented in this paper.

The second source of information is the Kilobotics website, and is a combination of tutorials and web based compiler that makes operation and programming of the units less difficult. The Kilobotics website has made improvements to the original process and therefore should be referenced before the *Kilobot Guide*. Before proceeding go the Kilobotics website and under the *Downloads* tab, download the Kilobot Bootloader (bootloader.hex), Controller Firmware (controller.hex) and the KiloGUI (kilogui.exe); place these files in a new folder inside the *System Software* folder and name it *Kilobotics Files* (if preferred kilogui.exe can be placed on the desktop for easy access).

Additions to Design Materials

The previous section explains all the materials one needs to get started making their own swarm. However, to complete these designs using the Kilobot Guide, Kilobotics, and the associated files only, can be a challenge. To combat the difficulties, some of the original documentation has been redone and some of the original procedures have been changed slightly. The changes in procedure will be described in this paper as needed. All document changes, corrections, or updates will be listed below. These documents should replace their original counterparts in the *DESIGN FILES* folders.

The original bill of materials (BOM) for the OHC and Kilobot units are located in the folders *OHC PCB* and *Robot PCB*, respectively, which are located in the folder *DESIGN FILES* inside the main directory of *Kilobot_documents*. Changes to the original documents are minimum; mostly done for consistency and readability. The improved BOMs can be seen in Tables 4 and 5.

The folder, *Robot PCB*, contains three more documents in need of an update. The documents *bottom_solder_mask_and_copper.pdf* and *top_solder_mask_and_copper.pdf* are the bottom and top PCB layouts for the Kilobots. These documents are used for component placement during soldering. The originals are crude and may contain errors; an improvement was made to these files (seen in figures 36 and 37).

Note: As shown in the updated PCB documents, what is considered the bottom of the PCB is actually the top of the robot and vice-versa.

The last document to address in this section, also contained in the *Robot PCB* folder, is the Kilobot Schematic labeled *Schematic.pdf*. This document is not at all legible and therefore, has been updated using Eagle Cad (seen in figure 38 and 39).

Parts Procurement

All of the electronic components for the Kilobot units and OHC can be obtained through a small number of distributors (see table 1). There may be a few parts on lengthy back order or obsolete by manufacturer; in this case different parts with the same specifications must be selected. To assist in this, the BOM includes descriptions of all components that contains rating and package information (package refers to the component footprint). Prices on these components can vary with time and between distributors. Extra money can be saved by researching the best price and price breaks for a given component through different distributors.

Table 1: Distributor List

Distributors Used	
Distributor	***Website***
Digikey	`www.digikey.com`
Mouser	`www.mouser.com`
Pololu	`www.pololu.com`
Powerstream	`www.powerstream.com`
Other Distributors	
Distributor	***Website***
Sparkfun	`www.sparkfun.com`
Ebay	`www.ebay.com`
Jameco	`www.jameco.com`
Adafruit	`www.adafruit.com`

PCB Design

A PCB design is contained in several CAM files. These files all have a different file extension to represent each layer of the PCB; table 2 shows common extensions[4]. The CAM files for the Kilobot units and the OHC are located in folders *Robot PCB* and *OHC PCB* respectively. Table 3 shows the CAM files contained in these folders. Also, in the *OHC PCB* folder are .Pcblib, .PrjPCB, .Schlib, .SchDoc, and .PcbDoc files exclusive to the PCB software suite Altium and are the schematic and PCB files that the CAMS were created from. These files are only given for the OHC and gives the user complete freedom to manipulate the PCB as desired. The CAM files are what gets sent to the PCB manufacturer. In this case OSH Park was used, although there are many other good choices.

Note: OSH Park is a reliable company with quick turn around and prices as cheap as \$1 per in^2 with quantity price breaks.

Any PCBs ordered, no matter the company come in multiples; this is why the quantity number in the BOM is higher than the number of units being built. To submit the CAM files to OSH Park they must be compiled in a compressed folder per PCB and submitted to the website. Before this is done a slight change must be made to file *Kilobot_ Controller_ PCB.GM1*; the extension must be changed from .GM1 to .GKO. After submitting the compressed folder, any errors will be addressed; if none, the PCBs can be ordered and usually arrive in about one to three weeks.

Table 2: OSH Park CAM Extensions

Ext.	**Layer**
.GTL	Top Layer
.GBL	Bottom Layer
.GTS	Top Soldermask
.GBS	Bottom Soldermask
.GTO	Top Silkscreen
.GBO	Bottom Silkscreen
.GKO	Board Outline
.TXT	NC Drill

Table 3: Project CAMS

OHC CAMS	**Kilobot CAMS**
Kilobot_Controller_PCB.GBL	BottomCopper.GBL
Kilobot_Controller_PCB.GBS	BottomSolder.GBS
Kilobot_Controller_PCB.GM1	BoardShape.GKO
Kilobot_Controller_PCB.GTL	TopCopper.GTL
Kilobot_Controller_PCB.GTO	
Kilobot_Controller_PCB.GTS	TopSolder.GTS
Kilobot_Controller_PCB.TXT	NCdrill.TXT

In addition to the PCBs, a Kilobot stencil for soldering the surface mount components (SMD) should be purchased as well. A stencil assists in applying the correct amount of solder paste on all the SMD pads. After the application of the solder and placement of parts, the entire PCB can be baked in a reflow oven, soldering all of the SMD components at once. OSH Stencil is a company that makes stencils from the top soldermask layer CAM file (.GTS)[5]. This file can be uploaded to the website and the stencil will arrive in one to two weeks.

Note: A PCB stencil is usually made with the paste mask CAM file; this is because only the surface mount components should be stenciled. By using the solder mask CAM, (the design files do not contain a paste mask CAM file) all parts, including vias, will be cut into the stencil. Because of this, care must be taken when applying the solder paste; only placing it on surface mount pads. Also, if a reflow oven is not available, a Kilobot can be hand soldered by and experienced individual.

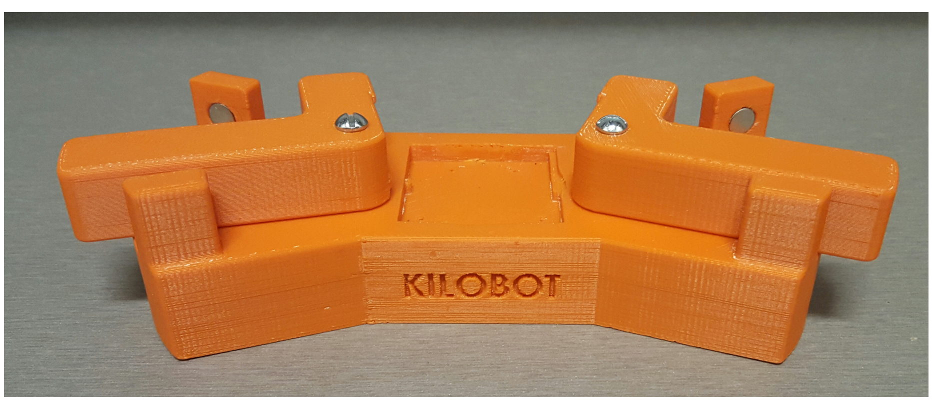
Kilobot Jig Assembly

Figure 4: Magnet Placement

1. Convert the three .SLDPRT jig files to .stl files so they can be 3D printed. Because .SLDPRT is a SolidWorks part file, it can be converted easily in SolidWorks. If SolidWorks is not accessible, Grabcad Workbench has a free web based converter that only requires the user to create an account[6].
2. Superglue 0.25" Neodymium magnets into the 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[2].
3. Screw in the two arms so they are firmly attached to the base but can still pivot[2].

   Note: The PCB may not lay flat inside the square opening of the jig. During the making of the shown jig the square had to be sanded out with a rotary tool so the PCB could be positioned correctly.

Figure 5: Kilobot Jig

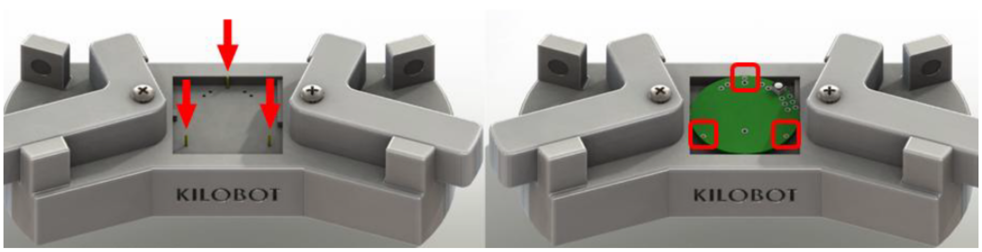


Kilobot Assembly

1. Using the solder stencil, apply an even layer of solder paste to the SMD pads on the top side of the PCB and, using figure 37 for reference, place the SMD components to the PCB making sure the infrared transmitter and receiver are pressed flat to the board surface (DO NOT scratch lens). If not pressed flat, the robot may transmit more light in one direction than in others, or be more sensitive to light in one direction; neither of which is desirable[2]. Repeat this step as to have enough boards to fill the reflow oven and bake them. It is important that the datasheet for the infrared components be reviewed and set the reflow oven profile so not to over heat these components. After the boards are removed from the oven hand solder the SMD components on the bottom of the boards (top of unit). All SMD components should be soldered before moving to the next step.

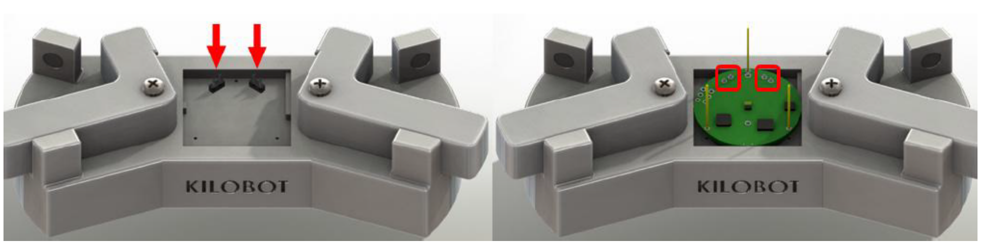
2. Insert three leg pins into the leg pin jig as shown (remove plastic from leg headers), pushing them down until they touch the table. Place the PCB in the jig as shown and solder the legs in place (the RGB led should be facing up)[2].

Figure 6: Leg Assembly



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[2].

Figure 7: Header Assembly

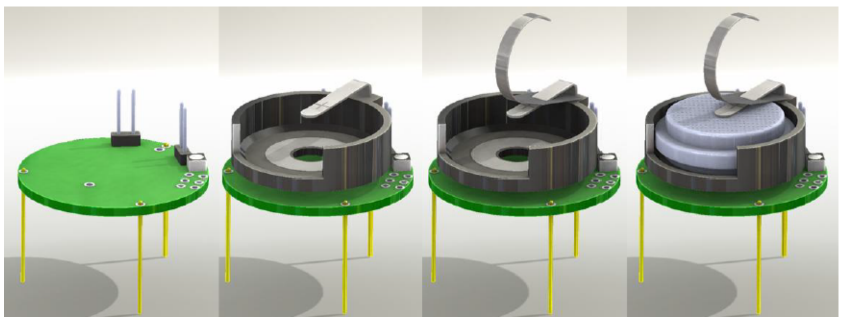


4. If needed, cut the motor leads to be 1.1" (28 mm) in length, strip and tin the tips, and solder onto the PCB using figure 36 as reference. Ensure that the motor leads do not cause any shorts[2].

   Note: Figure 36 is mirrored from what is seen looking down on the board and the red wire of the vibration motor should be connected to terminal labeled **+** and the blue wire to **-** shown in figure 36. This wiring assumes that the vibration motor will rotate in the clockwise direction with normal polarity (blue lead to ground). The left motor, *M2*, is connected with normal polarity and the right motor, *M1*, is reversed. This is why *M1's* postive lead is acctually connected to ground. The theory behind this type of locomotion platform is explained in more detail in *Analysis, Design and Control of a Planar Micro-robot Driven by Two Centripetal-Force Actuators*[7].

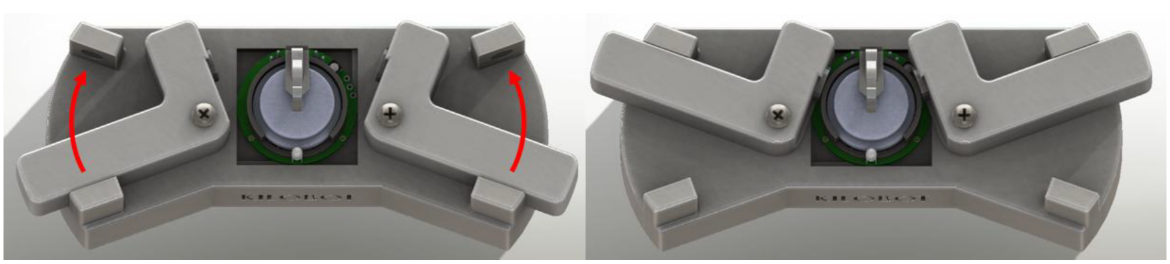
5. Solder the battery clip into the PCB making sure the motor wires are not tangled or pinched and insert a coin-cell battery **negative side up** as shown below (the charging clip seen below is optional and dimensions can be found in the Kilobot Userguide)[2].

Figure 8: Kilobot Assembly



6. Insert the robot into the motor assembly jig as shown below. 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. Quickly and carefully apply a small amount of hot glue to the underside of each motor and rotate the arms into place (glue can dry fast; make sure to close jig arms as soon as the glue is placed). Allow 30 seconds for the hot glue to cure.

Figure 9: Kilobot Assembly



7. 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 in 10[2]. The Kilobot is now complete.

Figure 10: Light Detector

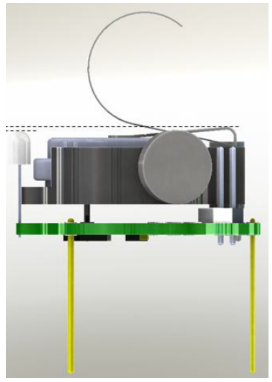
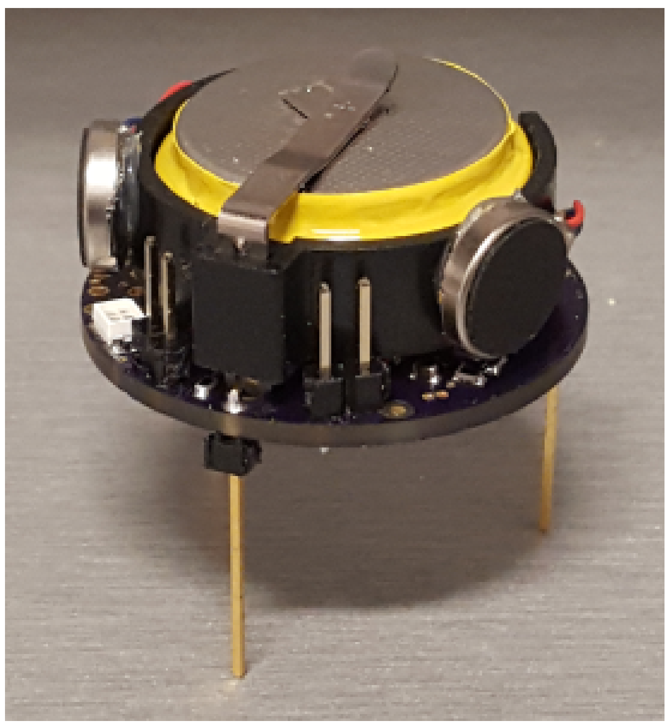


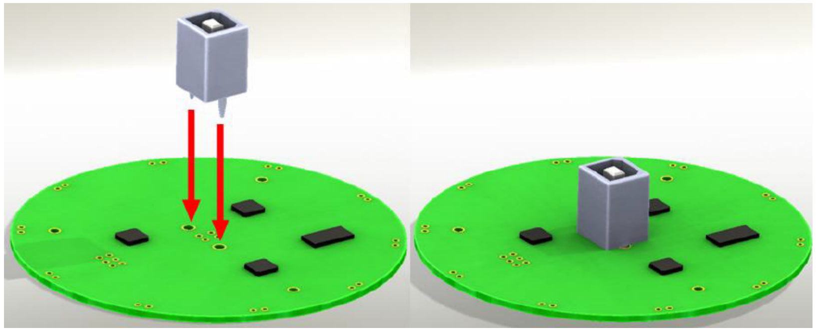
Figure 11: Complete Kilobot



OHC Assembly

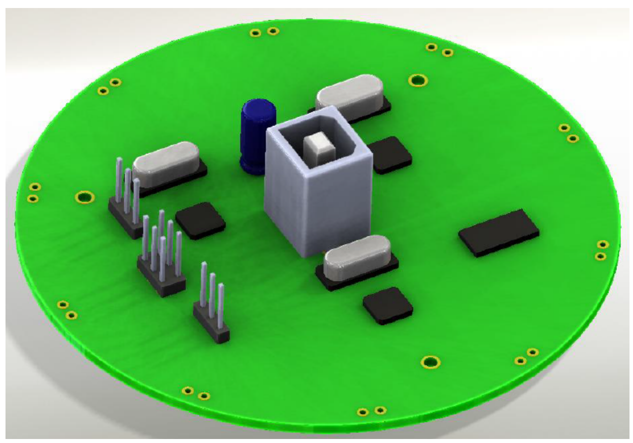
1. Using the silkscreen layer (component designation printed on the PCB), the part placement can be done by referencing the part designator in the BOM. The diodes and LEDs have arrow symbols between the component pads pointing to the cathode terminal. Hand solder the SMD components.
2. Solder USB header to the top side of the board as shown[2].

Figure 12: OHC Assembly



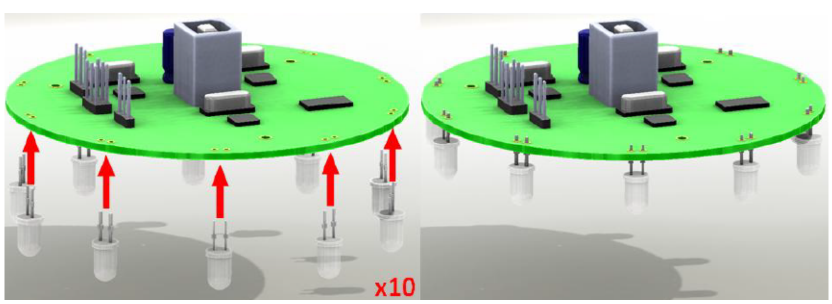
3. Solder the remaining top-side components as shown[2].

Figure 13: OHC Assembly



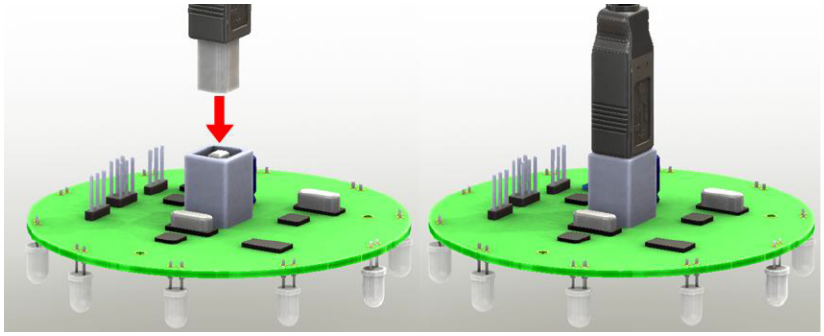
4. Solder the IR LEDs on the bottom side of the board as shown[2] (a straight line indicates the cathode terminal).

Figure 14: OHC Assembly



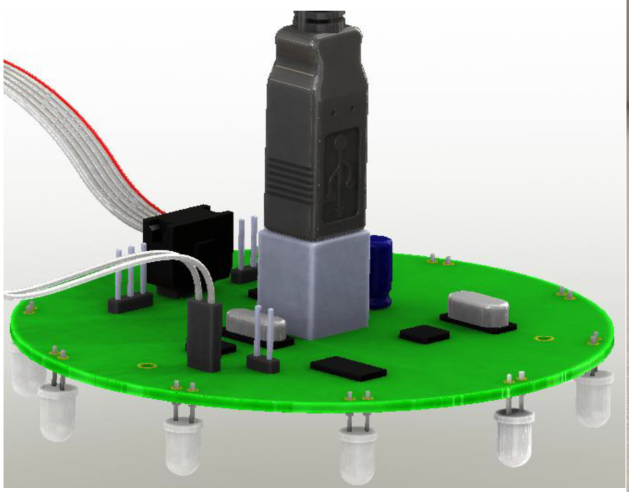
5. Insert the USB cable into the header as shown[2].

Figure 15: OHC Assembly



6. Attach kilobot programming cable and serial cable as shown. The programming cable is a 2x3 female header to 2x3 male header connected one to one (use the OHC and Kilobot schematics for reference). 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[2].

(a) OHC Model



(b) Complete OHC

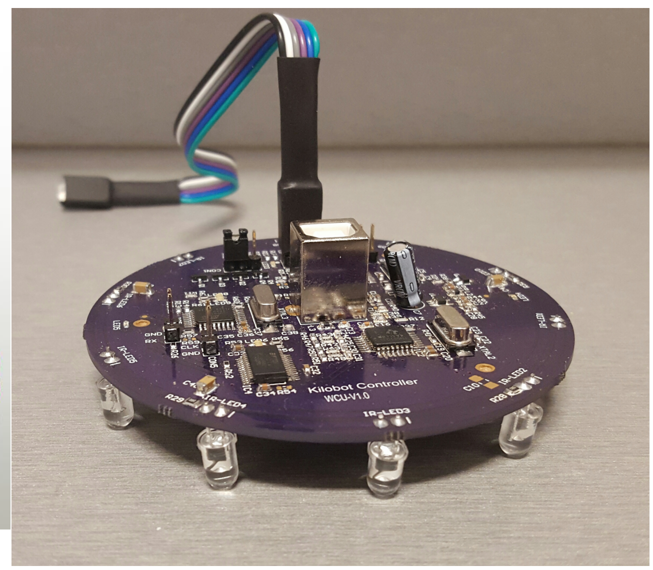


Figure 16: OHC Assembly

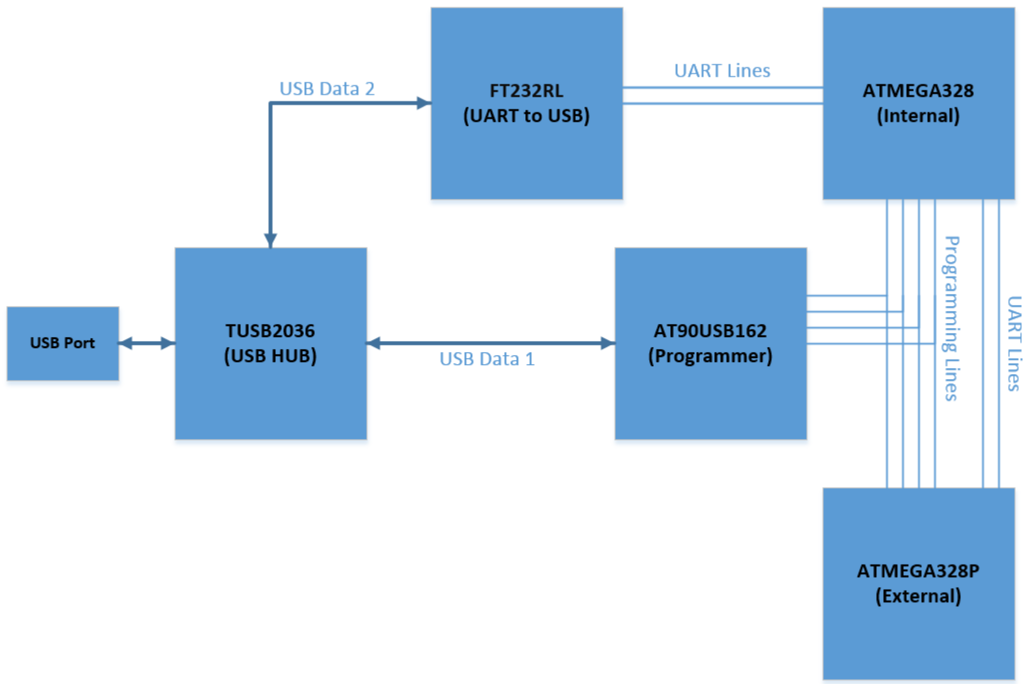
OHC Drivers and Firmware

The process about to be explained may have inconsistent results due to differences in operating systems and settings as well as software and drivers that may already be installed. This procedure is to assume that no needed drivers or software has been previously installed and the user's OS is Windows 7 (Windows 8 and 10 have not been tested with this procedure). Before starting it is recommended to prevent Windows from automatically installing drivers when a device is plugged in; this can be done in the *Device Installation Settings*.

Note: The diagram in figure 17 illustrates the connections between the ICs on the OHC. The diagram shows two USB data lines going into the a central hub then out to the PC. Because of this, there are several drivers needed:

1. The FT232RL driver. After installation of drivers, this connection will be seen as a COM port by the computer.
2. The AT90USB162 driver. After installation of drivers, this connection will be seen under Jungo with two tabs: WinDriver and AVRISP MKII.

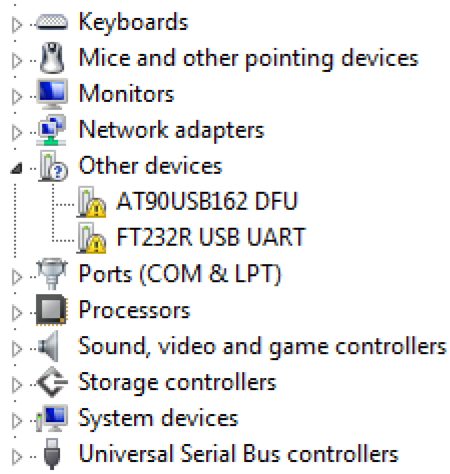
Figure 17: OHC Connection Diagram



Driver Procedure:

1. Install *AVR Studio 4.17 (build 666)*, *AVR Studio 4.18 SP1 (build 692)*, and *AVR Studio 4.18 SP3 (build 716)*[8] as well as WinAVR[9] (C and C++ compiler will be needed if AVR Studio will be used to compile code). AVR Studio 4 was chosen because it is easy to use and fast compared to Atmel Studio 6 which uses Visual Studio making it a large install that runs slow. AVR Studio 4 will be used later on for flashing the bootloaders on to the devices and also contains needed drivers.
2. Install *Atmel FLIP 3.4.7*[10]. This software will be used to program the OHC as an AVRISP MKII programmer and also contains needed drivers.
3. Plug the OHC into the computer (if drivers are automatically installed it will change the procedure slightly and certain steps may be skipped). The *Device Manager* should show the items below when the OHC is plugged in.

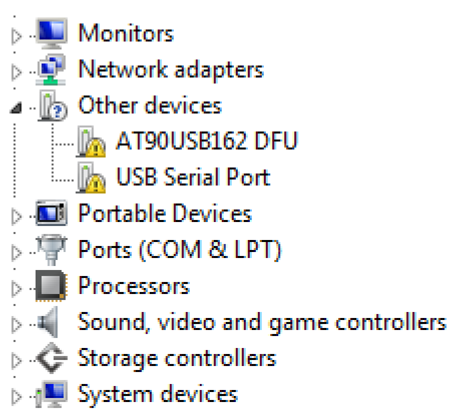
Figure 18: Device Manager



4. Copy the folder KilobotController (or just the relavent driver folder) to the main director of the C drive. Right click *FT232R USB UART* in the *Device Manager* 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[2].

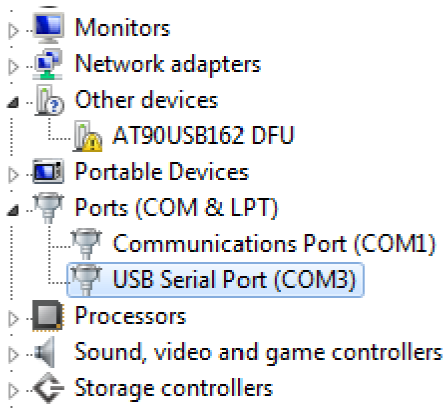
   Note: This driver is not final, and will be replaced. It is installed at this point to allow the user to use some of the older programs like *Kilobot Controller.exe* or some of the other calibration methods if desired. Doing it this way can also simplify the process.

   Figure 19: Device Manager

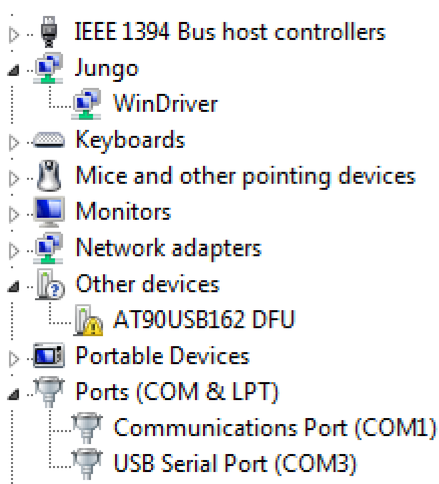
   If the *Device Manager* appears like figure 19, repeat this step by right clicking on *USB Serial Port*. When installed correctly the items should appear in the *Device Manager* as shown in figure 20.

Figure 20: Device Manager



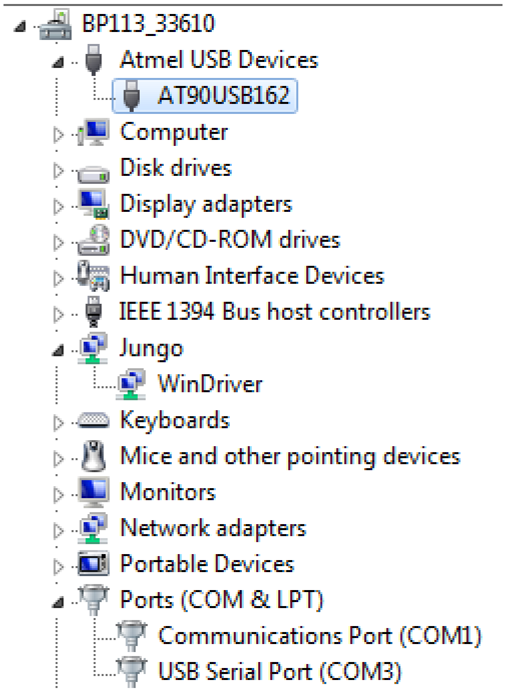
5. In the *Device Manager* at the top of the list, right click on the computer name and select *Add legacy Hardware*, select *Next*, *select Install the hardware, that I manually select from a List (Advanced)*, select *Show All Devices*, select *Have Disk*, point to *C:\Program Files (x86)\Atmel\AVR Tools\usb64\windrvr6.inf*, and lastly, select *WinDriver* an continue by selecting *Next* and *install* until complete. The *Device Manager* list should then appear as in figure 21.

Figure 21: Device Manager



6. Right click on *AT90USB162 DFU* and select *Update Drivers* and browse for *C:\Program Files (x86)\Atmel\Flip 3.4.7\usb* when complete, the *Device Manager* list should then appear as in figure 22.

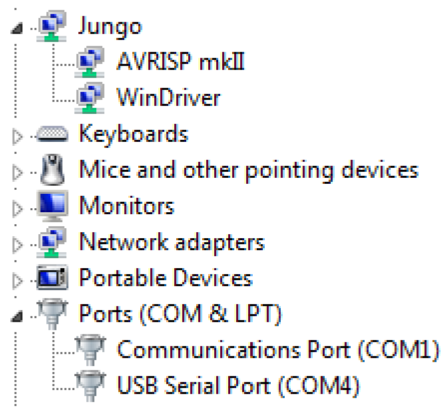
Figure 22: Device Manager



7. 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 the *AVRISP-MKII.hex* file located in the *System Software* folder. Check *Erase*, *Blank Check*, *Program* and *Verify*, then Push the *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 mk II* in the Device Manager under Jungo[2] as in figure 23. The OHC is now setup as explained in the *Kilobot Guide*. This means it is setup to work with the original GUI (Kilobot Controller.exe). If it is desired to use the original GUI and programs (not recommended) no more needs to be done to the OHC. Else, the final step to complete the OHC is in the following paragraph labeled ***IMPORTANT***.

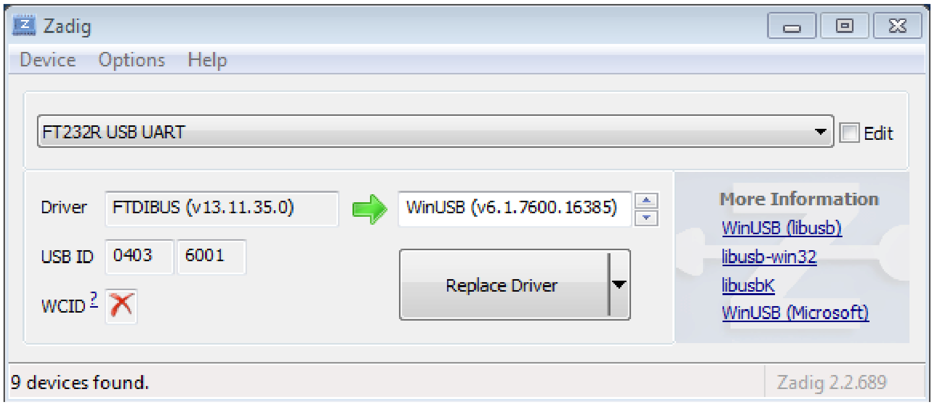
   Note: If this step does not work, refer to the *Hardware Boot Entrance Timing Characteristics* section of the *AT90USB162* datasheet . The IC may need a manual reset sequence before it will take the hex file; this is done by shorting R60 and R59 with a screwdriver in the proper order.

Figure 23: Device Manager



IMPORTANT: This material has been compiled assuming the files from the Kilobotics website will be used. One of these files is the *KiloGUI* (most recent GUI for controlling and programming the Kilobots) which will need specific drivers to work properly. Because windows may install drivers automatically during the previous procedure, it is important to insure that the correct drivers have been installed. A program called Zadig[11] allows the user to see what driver is currently installed for a specific device; it also makes replacing drivers easy. Download and run Zadig, go to options and select *List All Devices* in the pull down tab, select *FT232R USB UART*. The windows below the pull down tab will show the current driver on the left and to the right, possible drivers to install as seen in figure 24. *FTDIBUS(v13.11.35.0)* (located in the *FTDI64* folder inside *KilobotController* folder) is the driver for *Kilobot Controller.exe* (the original Kilobot GUI found in the *KilobotController* folder), which was installed in the previous steps and does not work well with *KiloGUI*, *WinUSB (v6.1.7600.16385)* is the driver that windows usually automatically installs, however *KiloGui* will not work with this driver either. *libusb-win32 (v1.2.6.0)* is the proper driver for the *KiloGui*. It is important to understand that the OHC will program through AVR Studio using any of these drivers and it is only KiloGUI that requires the use of *libusb-win32 (v1.2.6.0)* for correct operation. If the use of any of the older programs such as *Kilobot Controller.exe* is desired, the OHC driver should remain *FTDIBUS(v13.11.35.0)*. If the new programs (*KiloGUI.exe*)are to be used then the driver can be replaced with *libusb-win32 (v1.2.6.0)* using Zadig.

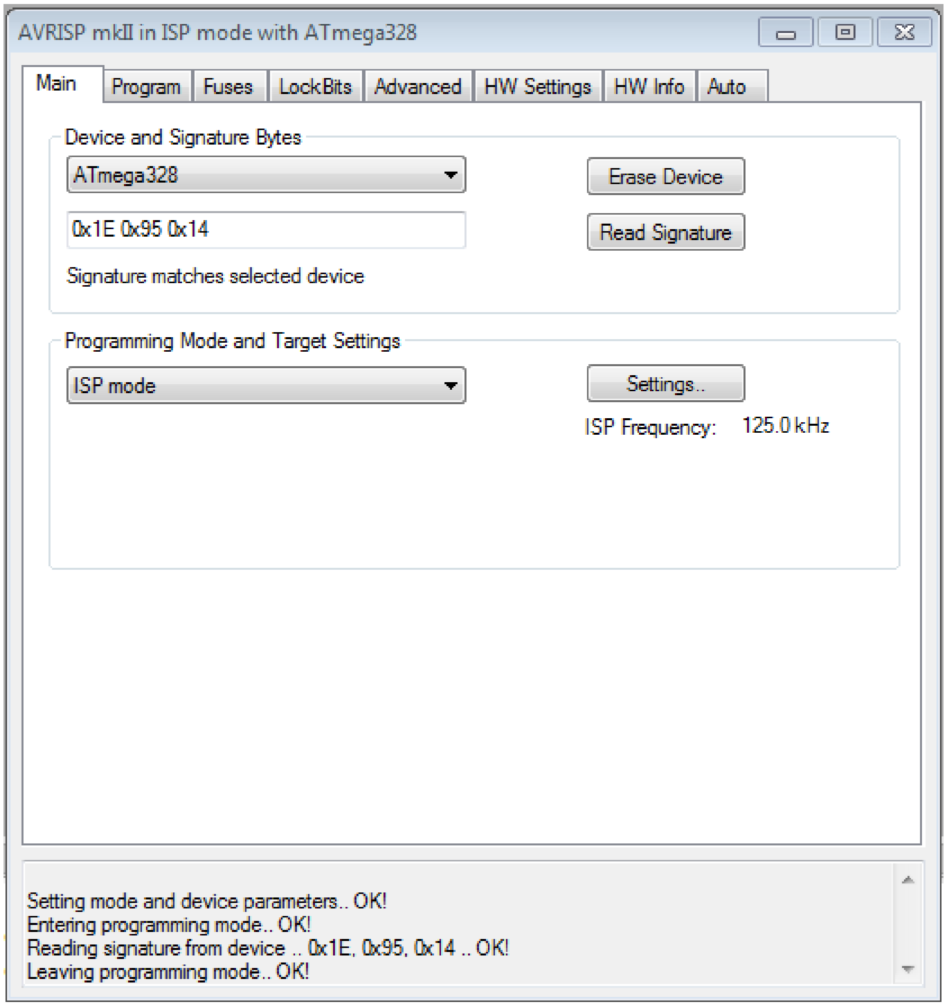
Figure 24: Zadig



Firmware Procedure:

1. Plug in the OHC (make sure the jumper on *CON1* is set to *INTERNAL PROG*) and open AVR studio.
2. Select *Tools / Program AVR / Connect...*
3. Select *AVRISP mkII* under *Platform:* and *USB* under *Port:* then select *Connect...*. If this window appears again, unplug the OHC, close the software, and repeat the previous steps.
4. Under the *Main* tab select *ATmega328* as the device. Click the *Read Signature* button, this should result in a Signature Code and several *OK!* at the bottom as seen in figure 25. This indicates that communication between the internal IC and the software is occurring correctly.

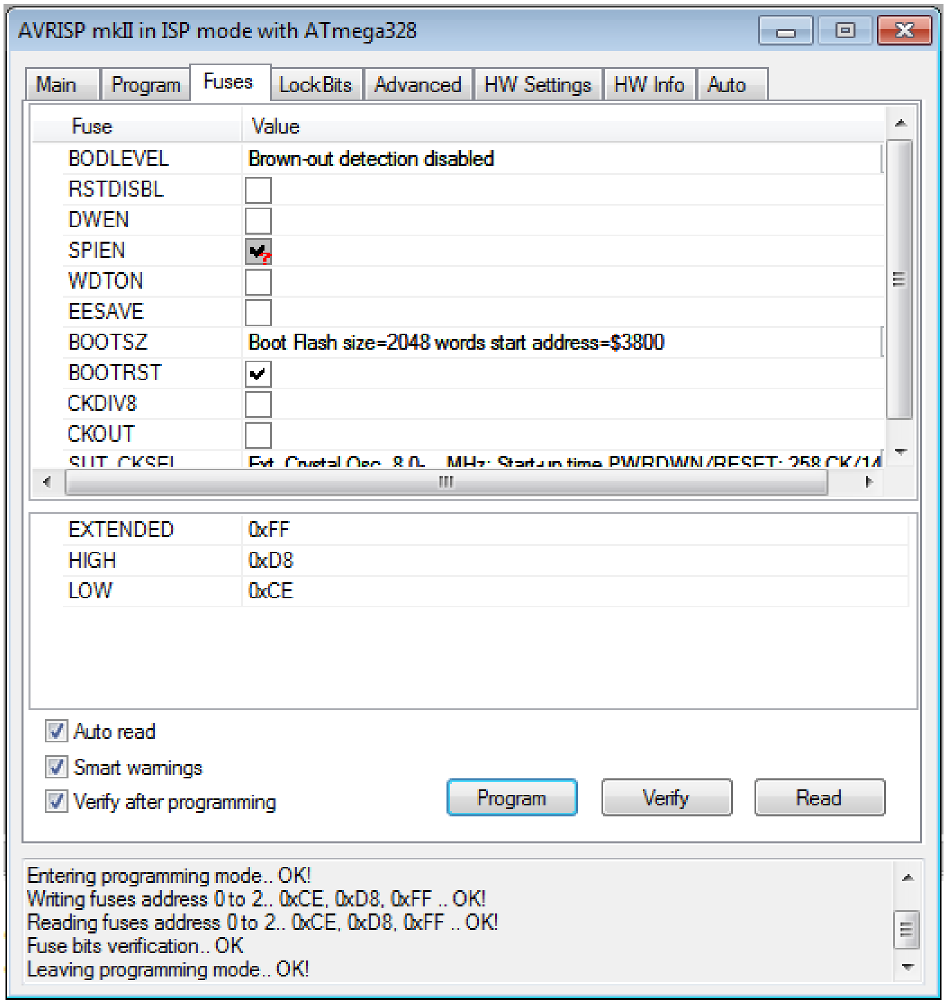
Figure 25: AVR Studio



5. Under the *Fuses* tab about mid page change the fuse values to EXTENDED = 0xFF, HIGH = 0xD8, and LOW = 0xCE. Then click *Program* and a verification will appear at the bottom similar to figure 26.

   IMPORTANT: Be extremely careful programming the fuse values. If incorrect values are entered and programmed all communication to the IC could be lost and the only practical way to recover from this is to replace the IC, so be mindful when changing these values.

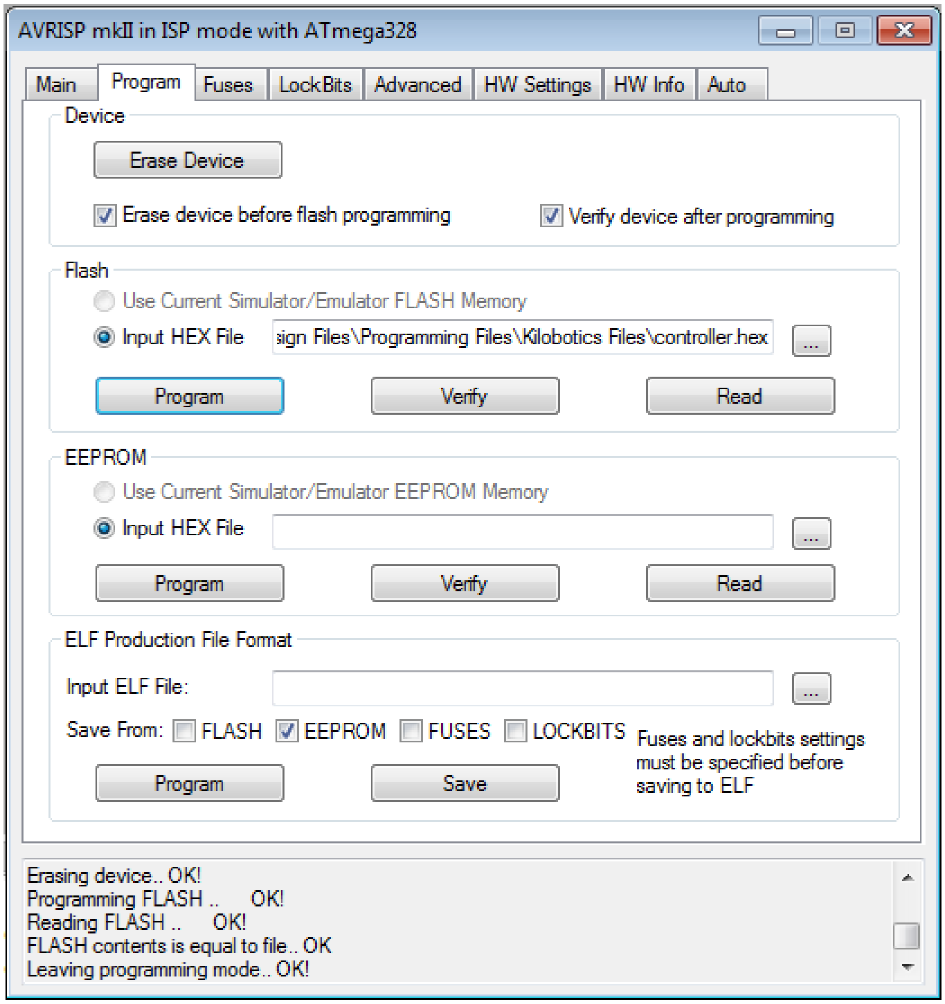
Figure 26: AVR Studio



6. Under the *Program* tab in the *Flash* section, browse for the *controller.hex* file that was placed in the *Kilobotics Files* folder earlier in this document. Click the *Program* button and another verification should appear at the bottom similar to figure 27.

7. The OHC is now ready to use.

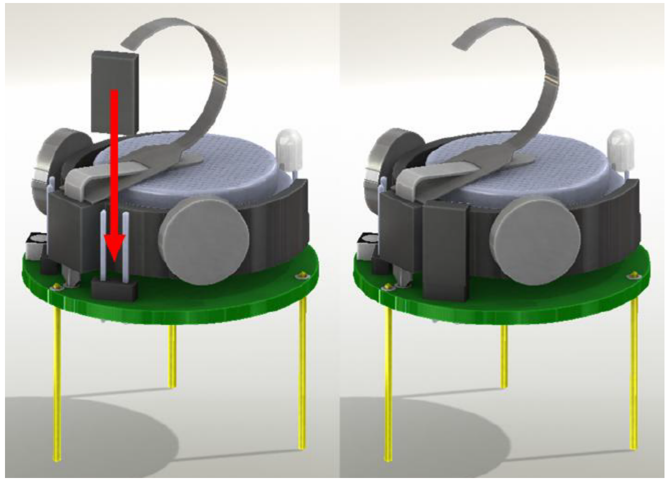
Figure 27: AVR Studio



Kilobot Bootloader

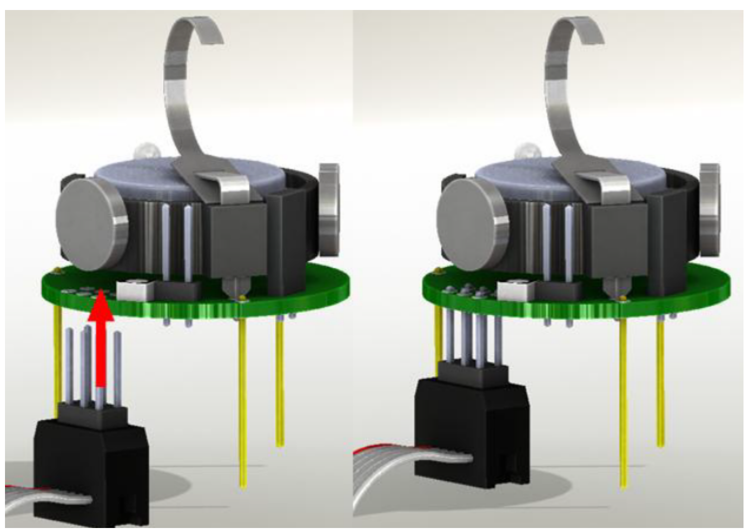
1. Plug in the OHC (make sure the jumper on *CON1* is set to *EXTERNAL PROG*) and open AVR studio.
2. Select *Tools / Program AVR / Connect...*
3. Select *AVRISP mkII* under *Platform:* and *USB* under *Port:* then select *Connect....* If this window appears again unplug the OHC, close the software, and repeat the previous steps.
4. Turn on the robot by adding the power jumper as shown in figure 28[2].

Figure 28: Turning On Kilobot



5. Using a one to one, female to male cable, 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.

Figure 29: Programming Connection



6. Under the *Main* tab select *ATmega328P* as the device. Click the *Read Signature* button, this should result in a Signature Code and several *OK!* at the bottom as seen in figure 25. This indicates that communication between the Kilobots IC and the software is occurring correctly.
7. Under the *Fuses* tab, about mid page change the fuse values to EXTENDED = 0xFF, HIGH = 0xD1, and LOW = 0xE2. Then click *Program*, and a verification will appear at the bottom similar to figure 26.

8. Under the *Program* tab in the *Flash* section browse for the *bootloader.hex* file that was placed in the *Kilobotics Files* folder earlier in this document. Click the *Program* button and another verification should appear at the bottom similar to figure 27; the robot may vibrate.

   IMPORTANT: Be extremely careful during the programming process. If the programming is interrupted it could cause incorrect fuse values to be programmed and all communication to the IC could be lost.

9. Repeat this procedure for the remaining Kilobot units.

Calibration

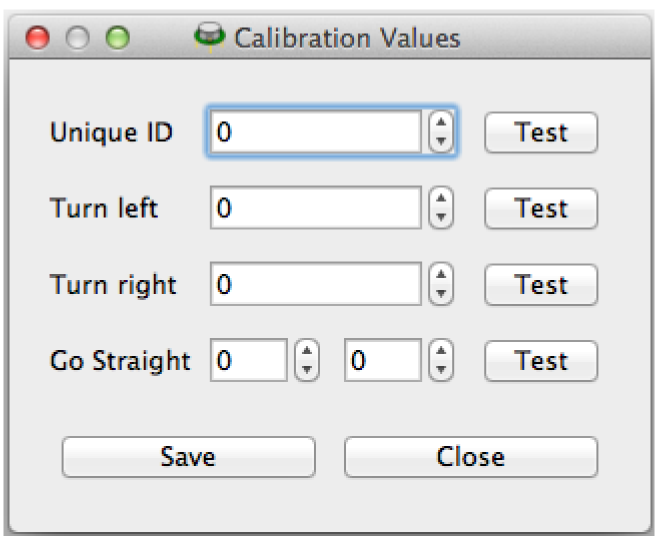
Motor Calibration:

Note: The procedure below is verbatim from the Kilobotics website unless labeled *Note:* which is additional information to the Kilobotic's procedure.

The Kilobots use vibration motors to move, this is known as stick-slip locomotion. Due to manufacturing differences the power required to achieve good forward and turning motion varies from robot to robot, and generally varies from surface to surface. Here is how to manually calibrate the values required for turning left, turning right, and going straight, in the process, you can also assign a unique identifier ("UID") to your kilobot, if you so desire[3].

1. Open up the KiloGUI program and click on the Calibration button, you will be presented with the following screen[3].

Figure 30: Motor Calibration



2. Select a value for turning left, click *test* to tell the robot to move using this value. Values between 60 and 75 work best for turning, but this will depend on your robot and on the surface being used. Choose different values until your robot can perform a full turn consistently on the surface being used[3].

3. Follow the same procedure for turn right. To calibrate moving straight, you can use the values you already found for turn left and turn right as a good initial guess. Usually go straight values should be between 2 and 10 units smaller than the turning left and turning right values to achieve a good motion[3].

4. Give your robot an identifier number by typing a positive integer in the unique ID box and clicking test. The ID can store an unsigned 16-bit integer but we recommend picking values that make sense given the size of the swarm you are using[3].

5. Once you have calibrated all the values, make sure to click SAVE to write these changes to the EEPROM memory of your robot. You will be able to use these values in your program, and this is described in the Kilobot Library, API docs[3].

   Note: During this procedure, it is likely that some Kilobot units will move opposite of the commands given, for example: a right command might move left, a left command might move right and both could occur resulting in a forward command moving in reverse. To solve this problem, pull the relevant motor off the battery holder and re-glue it upside down or remove the battery holder and reverse the polarity by reversing the wiring. For example, if the left turn command results in a right turn, then flip the motor that controls the left turn (left motor). This is done because the direction of rotation for each motor may be different given the same wiring polarity.

Distance Calibration:

Important: The Kilobots measure light intensity from the infrared transmissions of other units to determine the distance between the two. Each Kilobot has two sets (high and low gain) of 15 light intensity values stored in the EEPROM (Electrically Erasable Programmable Read-Only Memory) of the microprocessor that represent distances from 0mm to 70mm in 5mm increments. In the *DESIGN FILES* folder there are CAM files and schematics for a Calibration Board. This board transmits signals from the 15 distances that are then received by a Kilobot unit and programmed into the memory. For this procedure to work, the calibration unit must be calibrated and then each kilobot unit must be calibrated one at a time (see the *Kilobot User Guide*[2]). To avoid the extra expense and these extra steps, an easier method to implement was devised. This method will use *KiloGui.exe* and the OHC to upload and execute programs; make sure the materials on using the OHC and KiloGui (found at www.kilobotics.com/documentation) are understood before proceeding with this process.

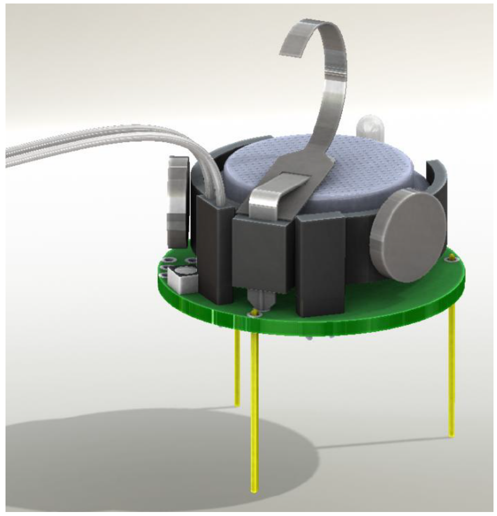
1. Using the Kilobotics editor[3], compile *RX_ CAL.c*, *TX_ CAL.c*, and *Distance_ Calibration.c*. These files can be found and copied from the end of this document. After compiling download and save the hex files in the *Kilobotics Files* folder.

2. Take three Kilobots with known working transmitters and receivers and label them A, B, and C.

3. Using the KiloGui and the OHC upload *RX_ CAL.hex* onto Kilobot A and *TX_ CAL.hex* onto Kilobot B. Connect a serial cable from the OHC to the serial header on unit A as seen in figure 31.

   Note: The ground connection for the debug wire is the pin located nearest to the front leg of the unit.

Figure 31: Serial Connection



4. Place units A and B so they are touching (0 mm), using the *KiloGui* execute by pressing *Run* and press the *Serial Input* button. Every time the RX_ CAL unit receives a signal the LED will flash yellow. After receiving ten signals the unit will output two values (low and high gain light intensities). These values are averages from the previous ten signals received. Repeat in 5 mm increments upto 70 mm. When all distance values have been documented there will be 15 high gain and 15 low gain values.

5. Repeat the previous three steps for every possible combination: B transmitting to A, A transmitting to C, C transmitting to A, etc. Collect the values in this manner until the data looks similar to figure 32.

   Note: If the receiving unit does not receive values at some of the longer distances leave the space blank.

Figure 32: Distance Calibration Data

	AB		BA		AC		CA		BC		CB	
Distance(mm)	**Low**	**High**	**Low**	**High**	**Low**	**High**	**Low**	**High**	**Low**	**High**	**Low**	**High**
0	488	1115	490	1125	469	1124	468	1125	519	1124	514	1115
5	399	1114	370	1125	348	1124	373	1125	431	1124	408	1114
10	304	1113	259	1125	306	1124	295	1125	326	1124	318	1114
15	222	1106	196	1023	220	1111	218	1109	254	1123	250	1113
20	174	927	148	818	185	972	166	888	196	1038	195	1032
25	124	695	124	679	148	784	138	755	156	832	157	846
30	100	557	96	551	102	568	107	600	119	645	119	659
35	73	423	70	403	81	455	81	465	91	509	95	528
40	57	354	52	327	60	339	61	364	66	387	76	460
45	39	250	36	238	48	281	46	298	48	293	58	354
50	30	200	26	172	33	218	34	225	37	245	47	286
55	16	123	16	147	25	175	28	207	26	185	34	223
60	14	116	9	103	13	118	19	132	15	131	26	170
65	10	97	7	95			11	93	6	87	16	119
70	8	87									8	90

6. Average all of the distance values and round to whole numbers. The final values in this case are shown in figure 33.

Figure 33: Final Distance Calibration Values

FINAL	
Low	**High**
491	1121
388	1121
301	1121
227	1098
177	946
141	765
107	597
82	464
62	372
46	286
35	224
24	177
16	128
10	98
8	89

7. Place the final values into the arrays labeled Cal_ Low and Cal_ High located in *Distance_ Calibration.c* .

Figure 34: Distance_ Calibration.c

```
//used locations in eeprom
#define ee_OSCCAL 0x001  // rc calibration value in eeprom, to be loaded to OSCCAL at startup
#define ee_SENSOR_LOW 0x20  //low gain sensor calibration data in epromm
#define ee_SENSOR_HIGH 0x50  //high-gain calibration data in epromm

void setup() {
        static int Cal_Low[15] = {491,388,301,227,177,141,107,82,62,46,35,24,16,10,8};
        static int Cal_High[15] = {1121,1121,1121,1098,946,765,597,464,372,286,224,117,128,98,89};
```

8. Compile *Distance_ Calibration.c* and download the hex file, save to the *Kilobotics Files* folder. Upload *Distance_ Calibration.hex* to all of the units to be calibrated and run. The LED will flash green for a few seconds and stop. The calibration values are now stored in the EEPROM of all the units.

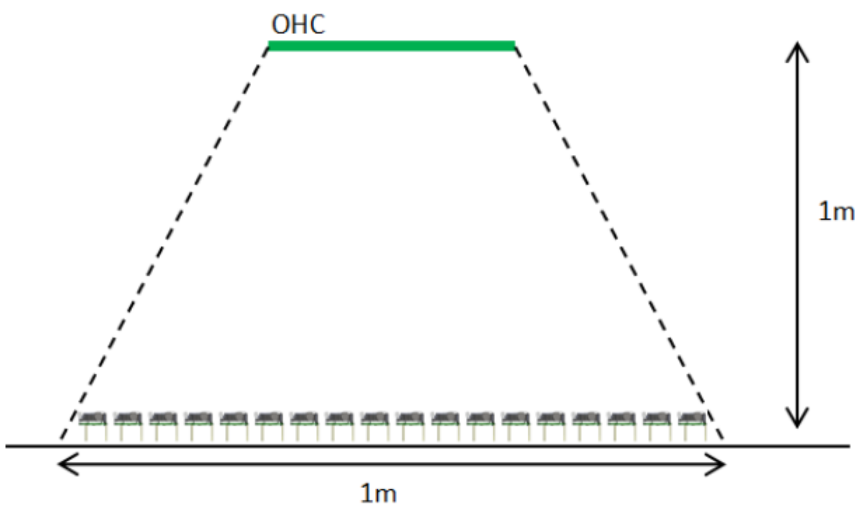
9. Program the unit as desired they are now ready for use.

   Note: The accuracy of this procedure depends on the surface used while collecting data, the consistency of the components and the quality of the manufacturing process. If any parts involving the IR sensors are substituted in a future batch of Kilobots or the arena surface changes the previous values collected may not be satisfactory.

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 about a one meter diameter region will be able to receive messages from the OHC as shown[2].

Figure 35: System Setup



Conclusion

The procedure presented in this document explains how to create a Kilobot Swarm. By following these steps one can build a large swarm with practical expenses. For comparison, a robotics company, K-Team, sells Kilobots and the supporting hardware like battery chargers and OHCs. K-Team sells a pack of 10 Kilobots for about $1100. The price for building units using these procedures comes out to be about $1400 which includes an OHC and 50 Kilobots. This latter figure means a research team willing to put in the work can have 100 units and support hardware under $3000.

Table 4: Kilobot BOM (50 Units)

Qty	Description	Distributor	Distributor #	Designation	Unit Price
450	CAP CER 2.2UF 6.3V Y5V 0603 (****)	Digikey	490-1587-1-ND	C1, C9, C10, C12, C14, C17, C18, C19, C20	0.0328
250	CAP CER 10000PF 25V 10% X7R 0603 (***)	Digikey	490-1520-1-ND	C3, C7, C8, C11, C13	0.00872
50	CAP CER 0.1UF 16V 10% X7R 0603 (**)	Digikey	478-1239-1-ND	C15	0.0094
50	CAP CER 1000PF 50V 5% NP0 0603 (*)	Digikey	490-1451-1-ND	C2	0.0244
50	CAP CER 10000PF 25V 5% C0G 0603 (**)	Digikey	445-2664-1-ND	C4	0.14
100	CAP CER 68PF 50V 5% C0G 0603 (**)	Digikey	445-1279-1-ND	C5, C6	0.0165
50	IC CHARGER LI-ION USB/AC 10WSON (*)	Digikey	LM3658SD-A/NOPBCT-ND	U6	1.1664
50	DIODE SCHOTTKY 30V 100MA 0603 (*)	Digikey	641-1282-1-ND	D1	0.2524
50	FIXED IND 10UH 50MA 900 MOHM (*)	Digikey	490-4025-1-ND	L1	0.101
50	IC OPAMP GP 5.5MHZ RRO 14TSSOP (*)	Digikey	296-22565-1-ND	U2	2.9232
50	RES SMD 0.82 OHM 1% 1/10W 0603 (*)	Digikey	P.82AJCT-ND	R38	0.1138
50	RES SMD 1.18K OHM 1% 1/10W 0603 (*)	Digikey	P1.18KHCT-ND	R13	0.015
50	RES SMD 1.47K OHM 1% 1/10W 0603 (*)	Digikey	P1.47KHCT-ND	R22	0.015
150	RES SMD 1.6K OHM 5% 1/10W 0402 (**)	Digikey	P1.6KJCT-ND	R25, R27, R29	0.0105
50	RES SMD 1.62K OHM 1% 1/10W 0603 (*)	Digikey	P1.62KHCT-ND	R3	0.015
150	RES SMD 10K OHM 1% 1/10W 0603 (**)	Digikey	P10.0KHCT-ND	R33, R34, R37	0.0114
50	RES SMD 10 OHM 1% 1/10W 0603 (*)	Digikey	P10.0HCT-ND	R16	0.015
50	RES SMD 11K OHM 1% 1/10W 0603 (*)	Digikey	P11.0KHCT-ND	R1	0.015
50	RES SMD 154 OHM 1% 1/10W 0603 (*)	Digikey	P154HCT-ND	R14	0.015
100	RES SMD 2M OHM 5% 1/10W 0402 (**)	Digikey	P2.0MJCT-ND	R31, R32	0.0105
50	RES SMD 2.26K OHM 1% 1/10W 0603 (*)	Digikey	P2.26KHCT-ND	R6	0.015
50	RES SMD 2.37K OHM 1% 1/10W 0603 (*)	Digikey	P2.37KHCT-ND	R11	0.015
50	RES SMD 20K OHM 1% 1/10W 0603 (*)	Digikey	P20.0KHCT-ND	R21	0.015
100	RES SMD 200 OHM 1% 1/10W 0603 (**)	Digikey	P200HCT-ND	R17, R24	0.0114
200	RES SMD 25.5K OHM 1% 1/10W 0603 (***)	Digikey	P25.5KHCT-ND	R5,R8, R10, R36	0.00844
50	RES SMD 280K OHM 1% 1/10W 0603 (*)	Digikey	P280KHCT-ND	R9	0.015
50	RES SMD 301 OHM 1% 1/10W 0603 (*)	Digikey	P301HCT-ND	R15	0.015
50	RES SMD 4.87K OHM 1% 1/10W 0603 (*)	Digikey	P4.87KHCT-ND	R20	0.015
50	RES SMD 499 OHM 1% 1/10W 0603 (*)	Digikey	P499HCT-ND	R7	0.015
50	RES SMD 4.99K OHM 1% 1/10W 0603 (*)	Digikey	P4.99KHCT-ND	R12	0.015
50	RES SMD 5.62K OHM 1% 1/10W 0603 (*)	Digikey	P5.62KHCT-ND	R2	0.015
50	RES SMD 549K OHM 1% 1/10W 0603 (*)	Digikey	P549KHCT-ND	R4	0.015
50	RES SMD 604K OHM 1% 1/10W 0603 (*)	Digikey	P604KHCT-ND	R35	0.015
150	RES SMD 820 OHM 5% 1/10W 0402 (**)	Digikey	P820JCT-ND	R26, R28, R30	0.0105
50	RES SMD 806 OHM 1% 1/10W 0603 (*)	Digikey	P806HCT-ND	R19	0.015
50	RES SMD 9.76K OHM 1% 1/10W 0603 (*)	Digikey	P9.76KHCT-ND	R18	0.015
	Not Mounted			R23	
50	IC REG LDO 3V 0.2A SOT25 (*)	Digikey	893-1100-1-ND	U3	0.3612
100	IC REG LDO 3V 0.7A SOT25 (**)	Digikey	893-1074-1-ND	U4, U5	0.4726
50	LED RGB 4PLCC (*)	Digikey	VAOS-SP4RGB4CT-ND	LED1	1.53
100	BJT NPN 20V 0.5A (**)	Mouser	755-2SD2114KT146W	T2, T3	0.129
150	TRANS NPN 50V 0.5A SOT-23 (**)	Digikey	MMBT6428CT-ND	T5, T6, T1	0.1353
50	MCU 32KB In-system Flash 20MHz (*)	Mouser	556-ATMEGA328P-MU	U1	2.41
100	Shaftless Vibration Motor 10x2.0mm (**)	Pololu	1638	M1, M2	2.62
50	Lithium ion Rechargeable Coin Cells (**)	Powerstream	Lir2477		2.02
50	SHUNT JUMPER .1" BLACK GOLD (*)	Digikey	3M9580-ND	Jumpers	0.06
3	CONN HEADER 50POS .100" SGL GOLD	Digikey	SAM1061-50-ND	J3, J4, J5	4.96
50	Coin Cell Battery Holder (*)	Mouser	534-1025-7	BATT	1.4
50	Infrared Emitters 5V 160mW 60Deg (*)	Mouser	782-VSMB1940X01	Tx1	0.735
50	Photodiodes 60V 215mW 60Deg (*)	Mouser	782-TEMD7100X01	Rx1	0.66
2	CONN HDR BRKWAY .100 80POS VERT	Digikey	A26536-40-ND	J1(POWER), J2(DEBUG),ISP1	3.54
50	Photo Transistor(Ambient Light Sensor) (*)	Mouser	782-TEPT5700	T4	0.478
90	PCB 1.7424 in^2 (Total Price = 157.50)	OSH Park			
					Total: 1282.73

* Price Break (50 Units)
** Price Break (100 Units)
*** Price Break (250 Units)
**** Price Break (500 Units)

Table 5: OHC BOM (1 unit)

Qty	Description	Distributor	Distributor #	Designation	Unit Price
12	CAP CER 0.1UF 25V 10% X7R 0603 (*)	Digikey	445-1316-1-ND	C1, C2, C8, C13, C15, C18, C19, C26, C29, C32, C34, C36	0.024
10	CAP CER 1UF 10V 10% X5R 0603 (*)	Digikey	490-1543-1-ND	C3, C6, C9, C14, C20, C27, C28, C33, C35, C37	0.041
12	CAP CER 22PF 50V 5% C0G 0603 (*)	Digikey	445-1273-1-ND	C4, C5, C16, C17, C22, C23, C24, C25, C30, C31, C38, C39	0.03
1	CAP ALUM 100UF 16V 20% RADIAL	Digikey	493-1040-ND	C7	0.25
2	47uf 10v (Not Mounted)	Digikey		C10, C12	
3	CAP CER 47UF 6.3V 20% X5R 1206	Digikey	490-3907-1-ND	C11, C40, C41	0.53
8	(Not Mounted)	Digikey		C21, R1, R3, R10, R40, R54, R59, R60	
1	CONN HEADER .100" SNGL STR 20POS	Digikey	S1011EC-20-ND	CON1,CON2,CON3,CON4,CON5	0.46
1	IC MCU 8BIT 32KB FLASH 32TQFP	Digikey	ATMEGA328-AURCT-ND	IC1	3.47
1	IC 2/3-PORT USB HUB 32-LQFP	Digikey	296-27129-1-ND	IC2	3.12
1	IC REG LDO 3.3V 0.15A SOT23-5	Digikey	TC2185-3.3VCCT-ND	IC3	0.5
1	IC USB FS SERIAL UART 28-SSOP	Digikey	768-1007-1-ND	IC4	4.5
1	IC MCU 8BIT 16KB FLASH 32TQFP	Digikey	AT90USB162-16AU-ND	IC5	4.35
10	EMITTER IR 870NM 100MA RADIAL (*)	Digikey	751-1208-ND	IR-LED1 to IR-LED10	0.939
3	FERRITE CHIP 1000 OHM 1.5A 0805	Digikey	445-5223-1-ND	L1, L2, L3	0.12
7	LED 572NM YW/GN WH/DIFF 0603 SMD	Digikey	511-1578-1-ND	LED1, LED2, LED3, LED6, LED7, LED8, LED9	0.54
2	LED BLUE HIGH BRIGHT ESS SMD	Digikey	LNJ937W8CRACT-ND	LED4, LED5	0.45
4	MOSFET N-CH 50V 220MA SOT-23-3	Digikey	BSS138KCT-ND	Q1, Q2, Q3, Q4	0.38
1	MOSFET N-CH 20V 2.1A SOT23-3	Digikey	ZXMN2B01FCT-ND	Q6	0.59
6	RES SMD 47K OHM 1% 1/10W 0603	Digikey	P47.0KHCT-ND	R2, R36, R52, R53, R57, R58	0.1
4	RES SMD 2K OHM 5% 1/10W 0603	Digikey	P2.0KGCT-ND	R4, R5, R6, R7	0.1
12	RES SMD 1K OHM 1% 1/10W 0603 (**)	Digikey	P1.00KHCT-ND	R8, R9, R12, R16, R17, R25, R26, R41, R55, R56, R61, R62	0.015
4	RES SMD 300 OHM 5% 1/10W 0603	Digikey	P300GCT-ND	R11, R13, R14, R15	0.1
7	RES SMD 28K OHM 1% 1/10W 0603	Digikey	P28.0KHCT-ND	R18, R19, R20, R21, R22, R23, R24	0.1
2	RES SMD 100 OHM 1% 1/10W 0603	Digikey	P100HCT-ND	R27, R34	0.1
5	RES SMD 1 OHM 5% 1/8W 0805 (*)	Digikey	RMCF0805JT1R00CT-ND	R28, R29, R30, R31, R32	0.021
1	RES SMD 1.5K OHM 5% 1/10W 0603	Digikey	P1.5KGCT-ND	R37	0.1
6	RES SMD 27 OHM 5% 1/10W 0603 (**)	Digikey	P27GCT-ND	R38, R39, R42, R43, R48, R49	0.0126
1	RES SMD 0.0 OHM JUMPER 1/10W	Digikey	RMCF0603ZT0R00CT-ND	R44	0.1
5	RES SMD 15K OHM 5% 1/10W 0603	Digikey	P15KGCT-ND	R45, R46, R47, R50, R51	0.1
1	CONN USB JACK TYPE B VERT STR	Digikey	151-1083-ND	USB1	2.1
1	CRYSTAL 8MHZ 18PF SMD	Digikey	535-10212-1-ND	XTAL1	0.35
1	CRYSTAL 6MHZ 20PF SMD	Digikey	XC1000CT-ND	XTAL2	0.56
1	CRYSTAL 16MHZ 18PF SMD	Digikey	535-10226-1-ND	XTAL3	0.35
3	PCB 11.4921 in^2 (Total Price = 57.55)	OSH Park			
					Total: 100.11

* Price Break (50 Units)
** Price Break (100 Units)
*** Price Break (250 Units)
**** Price Break (500 Units)

Figure 36: Bottom Layer (Top of Robot)

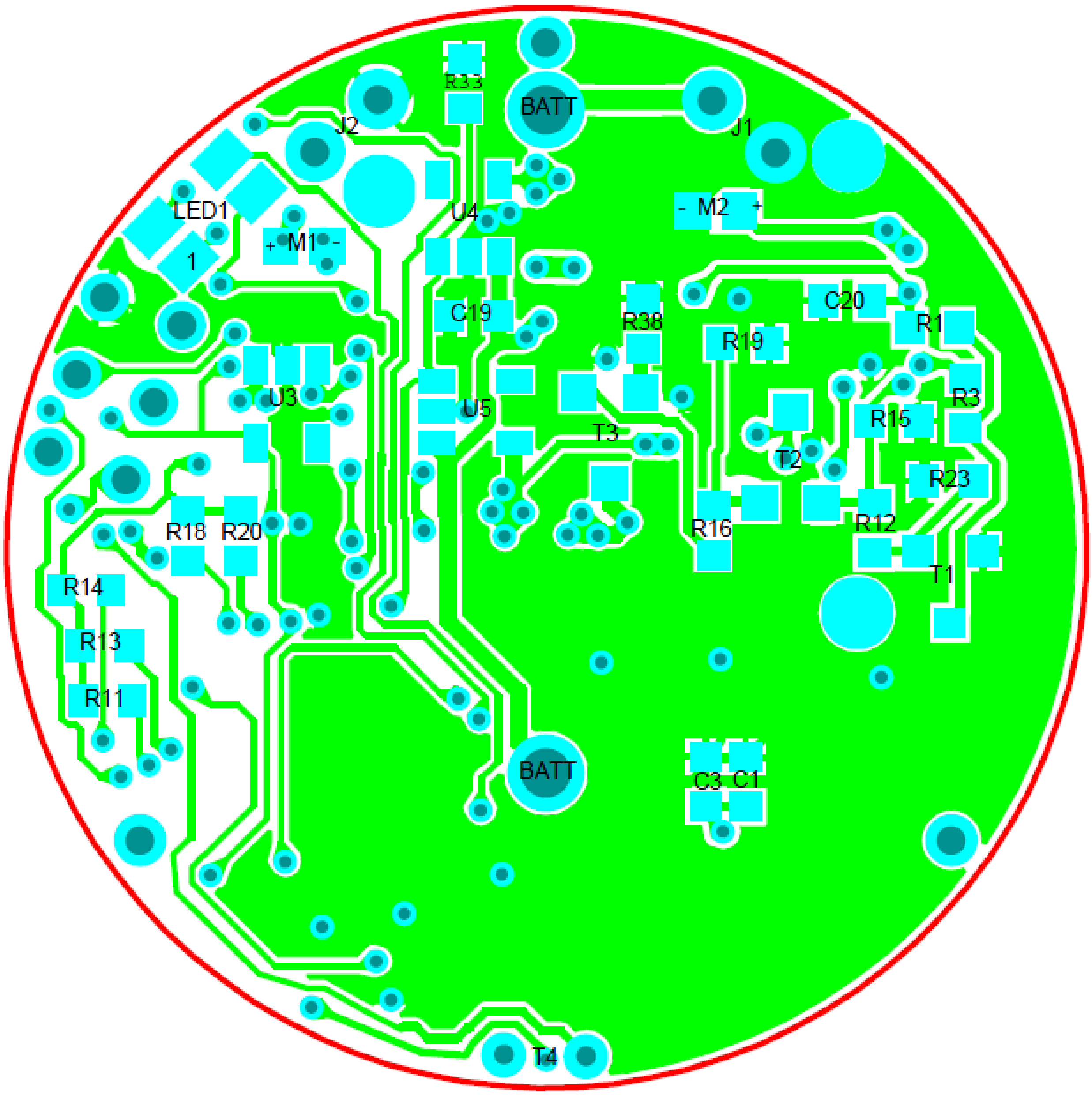


Figure 37: Top Layer (Bottom of Robot)

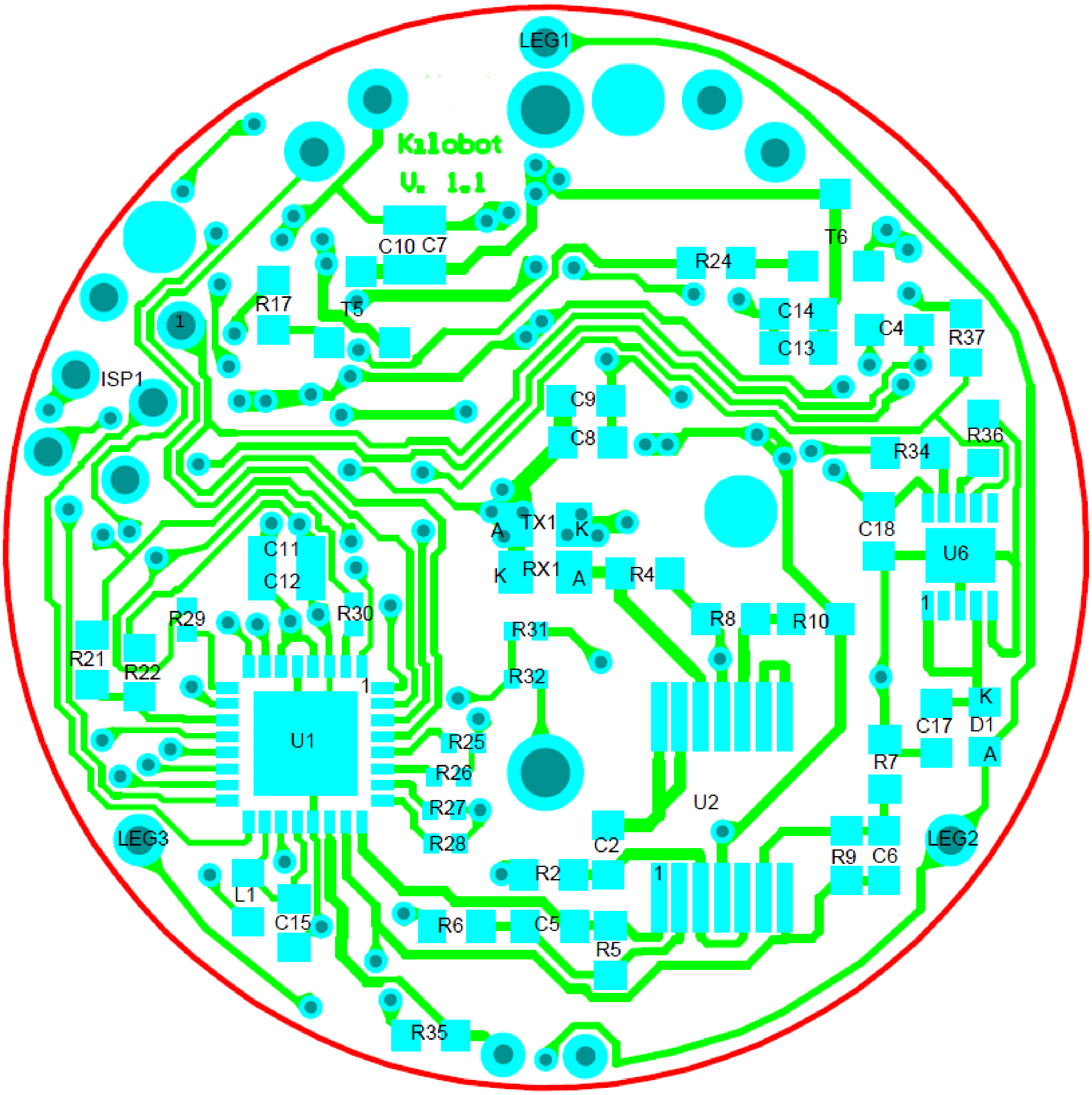
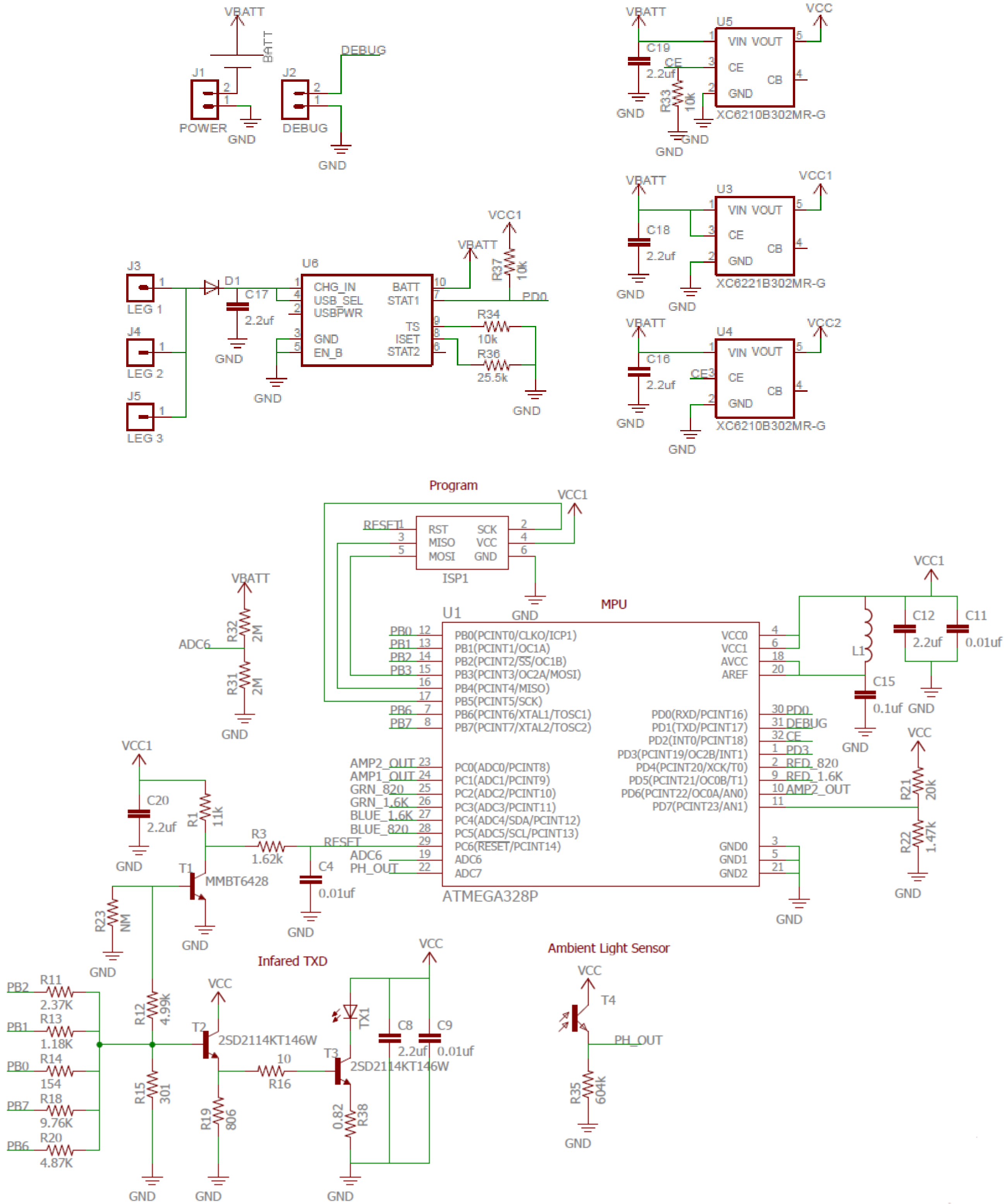


Figure 38: Kilobot Schematic



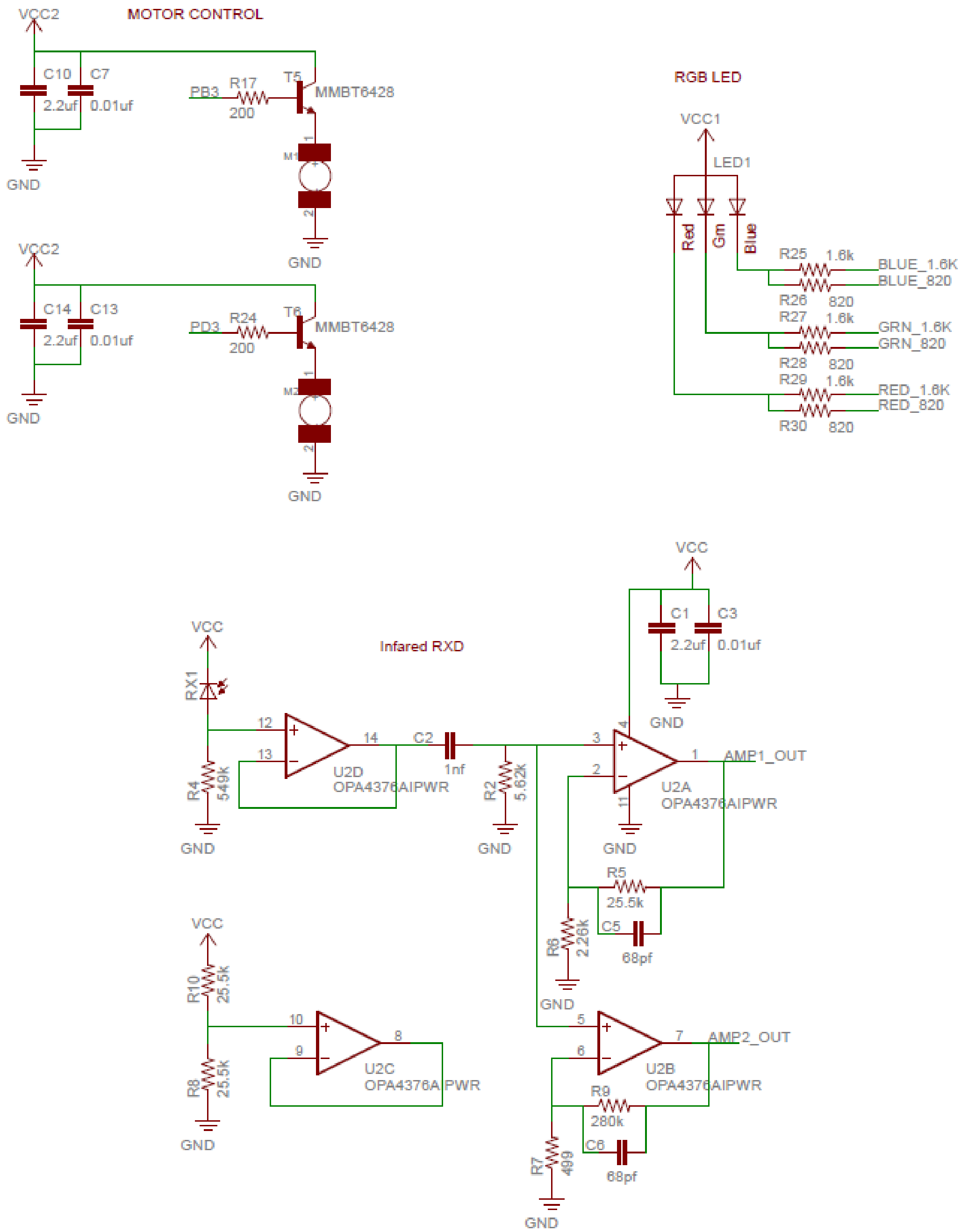


Figure 39: Kilobot Schematic Cont.

```
// RX_CAL.c

#include <kilolib.h>
#define DEBUG
#include <debug.h>

// Flag to keep track of new messages.
unsigned new_message = 0;
int lg;
int hg;
int lg_avg;
int hg_avg;
int lg_array[10];
int hg_array[10];
unsigned char i = 0;

void setup() { }
// print voltage every second
void loop() {

        // Blink the LED yellow whenever a message is received.
    if (new_message == 1)
    {
        // Reset the flag so the LED is only blinked once per message.
        new_message = 0;

        lg_array[i]=lg;
        hg_array[i]=hg;

        set_color(RGB(1, 1, 0));
        delay(100);
        set_color(RGB(0, 0, 0));
        i++;

        if (i == 10)
        {
         i = 0;
         lg_avg = (lg_array[0]+lg_array[0]+lg_array[1]+lg_array[2]+lg_array[3]
                  +lg_array[4]+lg_array[5]+lg_array[6]+lg_array[7]+lg_array[8]
                  +lg_array[9])/10;
         hg_avg = (hg_array[0]+hg_array[0]+hg_array[1]+hg_array[2]+hg_array[3]
                  +hg_array[4]+hg_array[5]+hg_array[6]+hg_array[7]+hg_array[8]
                  +hg_array[9])/10;
         printf("Low High\n");
         printf("%2d%6d\n", lg_avg, hg_avg);
        }
    }
}
void message_rx(message_t *message, distance_measurement_t *distance_measurement)
{
    new_message = 1;
    distance_measurement_t d;
    d = *distance_measurement;
    lg = d.low_gain;
```

```
    hg = d.high_gain;
}
int main()
{
    kilo_init();
    kilo_message_rx = message_rx;
    debug_init();
    kilo_start(setup, loop);
    return 0;
}
```

```c
// TX_CAL.c

#include <kilolib.h>

message_t message;
// Flag to keep track of message transmission.
int message_sent = 0;

void setup()
{
    // Initialize message:
    // The type is always NORMAL.
    message.type = NORMAL;
    // Some dummy data as an example.
    message.data[0] = 0;
    // It’s important that the CRC is computed after the data has been set;
    // otherwise it would be wrong and the message would be dropped by the
    // receiver.
    message.crc = message_crc(&message);
}

void loop()
{
    // Blink the LED magenta whenever a message is sent.
    if (message_sent == 1)
    {
        // Reset the flag so the LED is only blinked once per message.
        message_sent = 0;

        set_color(RGB(1, 0, 1));
        delay(100);
        set_color(RGB(0, 0, 0));
    }
}
message_t *message_tx()
{
    return &message;
}
void message_tx_success()
{
    // Set the flag on message transmission.
    message_sent = 1;
}
int main()
{
    kilo_init();
    // Register the message_tx callback function.
    kilo_message_tx = message_tx;
    // Register the message_tx_success callback function.
    kilo_message_tx_success = message_tx_success;
    kilo_start(setup, loop);

    return 0;
}
```

```c
// Distance_Calibration.c

#include "kilolib.h"
#include <avr/eeprom.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/sleep.h>

//used locations in eeprom
#define ee_OSCCAL 0x001  // rc calibration value in eeprom
#define ee_SENSOR_LOW 0x20  //Low-gain calibration data in epromm
#define ee_SENSOR_HIGH 0x50  //High-gain calibration data in epromm

void setup() {
    //High and low gain values to be placed in EEPROM
     static int Cal_Low[15] = {491,388,301,227,177,141,
                              107,82,62,46,35,24,16,10,8};
    static int Cal_High[15] = {1121,1121,1121,1098,946,765,
                              597,464,372,286,224,117,128,98,89};

    int i = 0;

while(i<=14)
{
set_color(RGB(0,1,0));
eeprom_write_byte((uint8_t *)(ee_SENSOR_LOW+i*2),((Cal_Low[i])>>8));
_delay_ms(100);
set_color(RGB(0,0,0));
eeprom_write_byte((uint8_t *)(ee_SENSOR_LOW+i*2+1),(Cal_Low[i] & 0x00ff));
_delay_ms(100);

set_color(RGB(0,1,0));
eeprom_write_byte((uint8_t *)(ee_SENSOR_HIGH+i*2),((Cal_High[i])>>8));
_delay_ms(100);
set_color(RGB(0,0,0));
eeprom_write_byte((uint8_t *)(ee_SENSOR_HIGH+i*2+1),(Cal_High[i] & 0x00ff));
_delay_ms(100);
i++;
}
}

void loop() {

}
int main() {
    // initialize hardware
    kilo_init();
    // start program
    kilo_start(setup, loop);

    return 0;
}
```

References

[1] Michael Rubenstein, Christian Ahler, and Nagpal Radhika. Kilobot: A low cost scalable robot system for collective behaviors. *In Proceedings of 2012 IEEE International Conference on Robotics and Automation (ICRA 2012): May 14-18*, pages 3293–3298, 2012.

[2] Radhika Nagpa. The kilobot project, 2008. URL http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html.

[3] Kilobotics, 2013. URL https://www.kilobotics.com/.

[4] Osh park design submission guidelines, 2016. URL https://oshpark.com/guidelines.

[5] Osh stencil, 2016. URL https://www.oshstencils.com/.

[6] Grabcab workbench, 2016. URL https://grabcad.com/workbench.

[7] Panagiotis Vartholomeos and Evangeloss Papadopoulo. Analysis, design and control of a planar micro-robot driven by two centripetal-force actuators. *In Proceedings of 2006 IEEE International Conference on Robotics and Automation (ICRA 2006): May 14-18*, pages 649–654, 2006.

[8] Atmel and avr studio, 2016. URL http://www.atmel.com/tools/STUDIOARCHIVE.aspx.

[9] Winavr, 2016. URL https://sourceforge.net/projects/winavr/files/WinAVR/20100110/.

[10] Atmel flip, 2016. URL http://www.atmel.com/tools/FLIP.aspx.

[11] zadig, 2016. URL http://zadig.akeo.ie/.