Botball 2013
Educators' Workshop

V1.4.1 2013.01.22

Day 1: Getting Started

1. Sign in and collect your materials and electronics
2. **Read the information on the next slide about charging a Link controller**
3. Reference workshop slides (Activity 0 ~ slide 28) for installation of current KIPR Link firmware and KISS IDE software
   - At your Team Home Base (on a flash drive at workshops without internet)
4. Update the firmware on your KIPR Link controllers
5. Install current version of the KISS IDE and KIPR Link USB driver for your computer (see Activity 0 for Mac and Windows specifics)
6. Go through the parts list and materials and verify everything is present
7. If you are not sure how to do any of the previous steps please ask one of the staff!

Charging the KIPR Link Controller

- For charging the KIPR Link, **use only the power supply which came with your Link**
  - **Damage to the Link from using the wrong charger is easily detected and will void your warranty!**
- The KIPR Link power pack is a lithium polymer battery so the rules for charging a lithium battery for any electronic device apply
  - You should NOT leave the unit unattended while charging
  - Charge away from any flammable materials and in a cool, open area

2013 Botball National Sponsors

Regional Workshop & Tournament Hosts

UNF UNIVERSITY of NORTH FLORIDA

TU WIEN TECHNISCHE UNIVERSITÄT WIEN Vienna University of Technology

SOUTHERN ILLINOIS UNIVERSITY EDWARDSVILLE

APL STEM JOHNS HOPKINS UNIVERSITY Applied Physics Laboratory

RICE Unconventional Wisdom

NASA

S·T·E·M Outreach A MITRE Corporation Initiative

WORCESTER STATE UNIVERSITY

NM STATE UNIVERSITY

UMASS LOWELL

HAWAI'I HAWAIʻI CONVENTION CENTER WHERE BUSINESS AND ALOHA MEET™

Georgia Tech College of Computing Computational Science and Engineering

جامعة كارنيجي ميلون في قطر

Carnegie Mellon Qatar

WizKidz Science and Technology Centers

University of San Diego

MESA Mathematics Engineering Science Achievement

UNIVERSITY OF MARYLAND

30 years COLORADO MESA Mathematics Engineering Science Achievement

THE UNIVERSITY OF ARIZONA. Arizona's First University.

MOREHOUSE COLLEGE PROJECT IDENTITY A Title III Program

CALIFORNIA STATE UNIVERSITY SAN BERNARDINO

Botball®

Botball 2013

prepared by:

The KISS Institute for Practical Robotics

with significant contributions from
the staff of KIPR and the Botball Instructors Summit participants

Housekeeping

Day 1

- Bathrooms, food
- Introductions
  - First time teams are identified by a colored name tag
    - If you’ve done this before, you know how they feel!
    - Please help them out
  - Workshop staff
- Daily schedule

Workshop Schedule

- Day 1:
  - Overview of Botball
    - Botball season, related events
    - Game preview/video
    - Resources & teams
  - Topics and Activities
    - Activity 0: The KISS IDE
    - Activity 1: Programming basics
    - Activity 2: Driving Straight
    - Activity 3: Build DemoBot
  - Lunch
    - Activity 4: Conditions and functions
    - Activity 5: Starting / shutting down the robot using sensors
    - Activity 6: Motors and servos
    - Activity 7: Line following
  - Homework

- Day 2:
  - New Team Suggestions
  - 30 minute game Q&A
  - BOPD
  - Continue with activities
  - Topics and Activities
    - Activity 8: Vision
  - Lunch
  - Selected activities
    - Activity 9: Point servo at colored object
    - Activity 10: Bang-Bang control
    - Activity 11: Proportional control
    - Activity 12: Approach specific QR code
    - Activity 13: Bang-Bang DemoBot arm
    - Activity 14: Proportional DemoBot arm
    - Activity 15: Accelerometer for bump detect
    - Activity 16: Music on the Create
    - Activity 17: Reduce heading errors

Overview of Botball

- Botball is brought to you by the KISS Institute for Practical Robotics, hereafter referred to as KIPR
- Botball season
- Game preview/video
- Botball Related events
  - Global Conference on Educational Robotics (GCER)
  - KIPR Open (for those beyond Botball)
  - KIPR Video Showcase
- Related curriculum topics

2013 Botball Season

Jan - Mar 2013

- Botball Professional Development Workshops
- KIPR Open Game released

February 2013

- GCER Call for Papers
- GCER Call for Autonomous Showcase submissions

Mar - May 2013

- Botball Regional Tournaments

April 2013

- ECER

Spring 2013

- GCER Registration Opens

July 2013

Global Conference on Educational Robotics (GCER)

- Fun and networking
- International Botball Tournament
- KIPR Open Tournament
- Autonomous Showcase
- Presentations/Papers/Guest Speakers

The Botball Youth Advisory Council

We are a group of current and former Botballers who form Botball's student government. We work on many projects (e.g. blogs, forums, live-streaming), with one simple mission: keep making Botball better!

Botball Community

community.botball.org

A place for Botballers

- Discuss Botball, technology, and everything else during and after the Botball season
- Contains a safe and user-friendly chat-room, the Botballer's Chat, for getting immediate help to technical problems, or just hanging out with fellow Botballers across the globe
- The Community is a social network for current and former Botball participants to meet and hang out, discuss Botball and robotics technology in general, or just have a good time

A Botball YAC and KIPR Project

ECER13 - Hard Facts

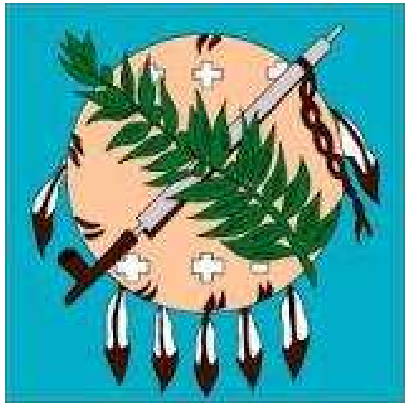
- European Conference on Educational Robotics
  - 2nd Botball competition in Europe
- Venue: Vienna, Austria
  - TGM (Vienna Institute of Technology)
  - Vienna Museum of Technology
- Two main parts:
  - European Botball competition
  - Talks of Researchers and Students

April
23th - 27th
2013

Botball®

GCER 2013

The 2013 Global Conference on Educational Robotics will be held at the Embassy Suites Hotel and Conference Center in Norman, Oklahoma from
July 6-10, 2013 with preconference classes on July 5th



Global Conference on Educational Robotics
http://www.kipr.org/gcer

Conference events will be held onsite in the conference facilities. We have secured a block of rooms at the Embassy Suites - when making reservations, refer to
http://www.kipr.org/gcer/housing
Rooms at the conference rate will be available until May 18.

Botball®

Global Conference on Educational Robotics

[ALL TEAMS ARE INVITED!]

When

– July 6th - July 10th
– Pre-conference activities and workshops July 5th

Who

– Middle school and high school students, educators, robotics enthusiasts, and professionals from around the world

Activities

– Meet and network with students from around the country and world
– Talks by internationally recognized robotics experts
– Teacher, student, and peer reviewed track sessions
– International Botball Tournament
– KIPR Open Tournament (Botball for grown-up kids!)
– Autonomous Robotics Showcase
– Visit America's heart land (conference rates!)

Botball®

Coming July, 2014

GCER 2014

Location and Date TBA

Global Conference on Educational Robotics

http://www.kipr.org/gcer

KIPR Open Tournament

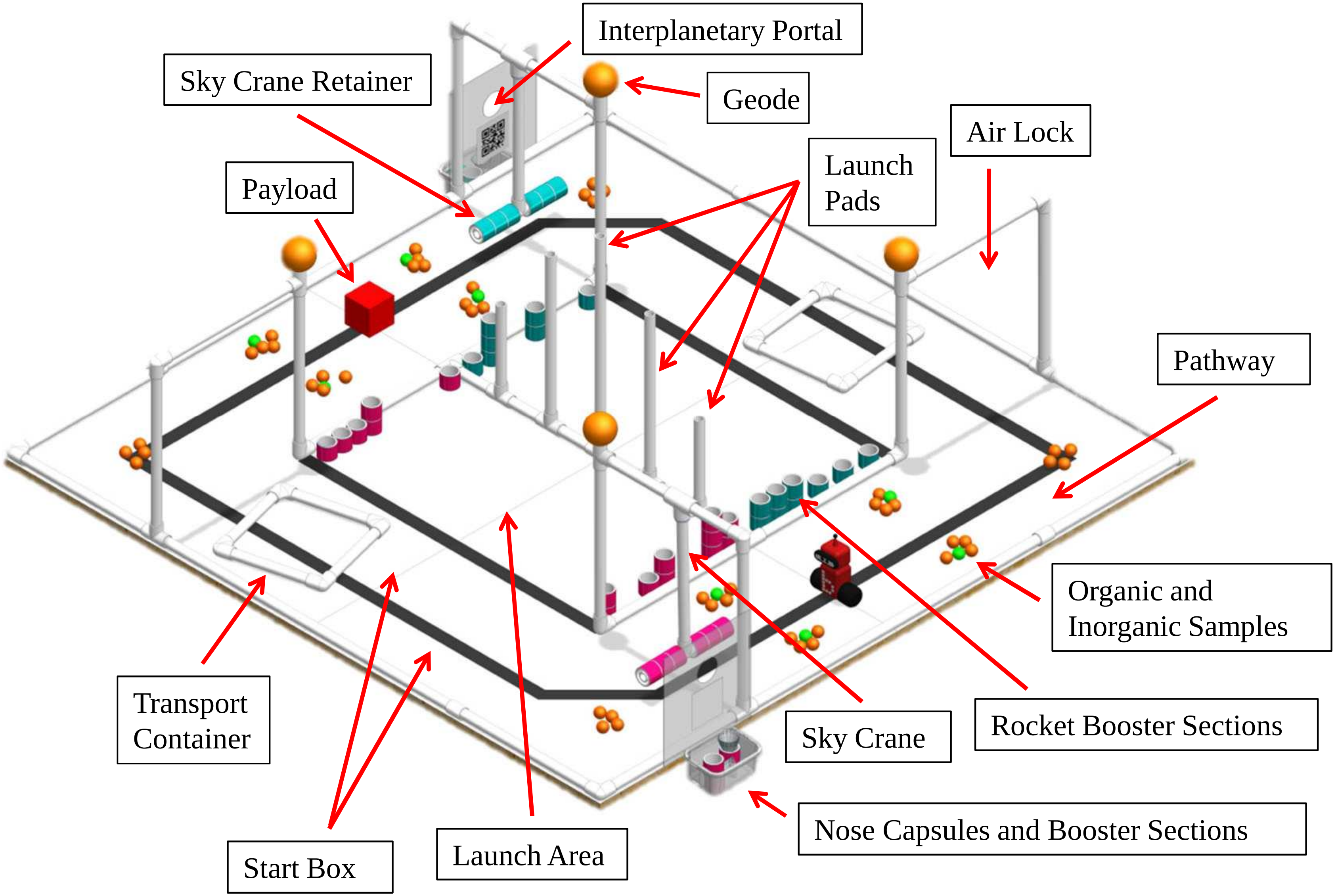
- The KIPR Open is a tournament produced by KIPR to encourage ongoing robotics educational activity beyond high school and Botball
- KIPR Open team entry forms and conference registration can be found at **www.kipr.org/kipr_open**
- The 2013 International KIPR Open Tournament will be held in conjunction with GCER 2013 July 6-10, 2013
  - See the KIPR website for information on KIPR Open Tournaments
- Collegiate courses are encouraged to incorporate the KIPR Open Game into their curriculum

Preview of This Year's Game

Hold Your Questions! Game Q&A is Tomorrow

The Game Board

Hold Your Questions! Complete Review is Tomorrow



Tonight-Review the Game Rules on Your Team Home Base

- We will have a 30 minute Q&A session tomorrow
- After the workshop, ask questions about game rules in the Game Rules Forum (the rules are at the Team Home Base)
  - You should regularly review this Forum
  - You will find answers to game questions there

Team Home Base Resources

(At homebase.kipr.org)

Resources!

- **On your Team Home Base**
  - Documentation Manual and examples
  - Presentation Rubric & Example Presentation
  - Demobot build instructions & Parts List
  - Controller Getting Started Manual
  - Construction Examples
  - Hints for New Teams
  - Sensor & Motor Manual
  - Game Table construction documents
  - All 2013 Game Documents
- **Botball Community Site** http://community.botball.org/
- **KISS Institute for Practical Robotics Tech Support** +1-405-579-4609

Why are we teaching this?

- There are enormous problems facing humanity today.
- There are also small, but important problems that no one has figured out how to solve.
- Today's students will need to approach these challenges in innovative ways.

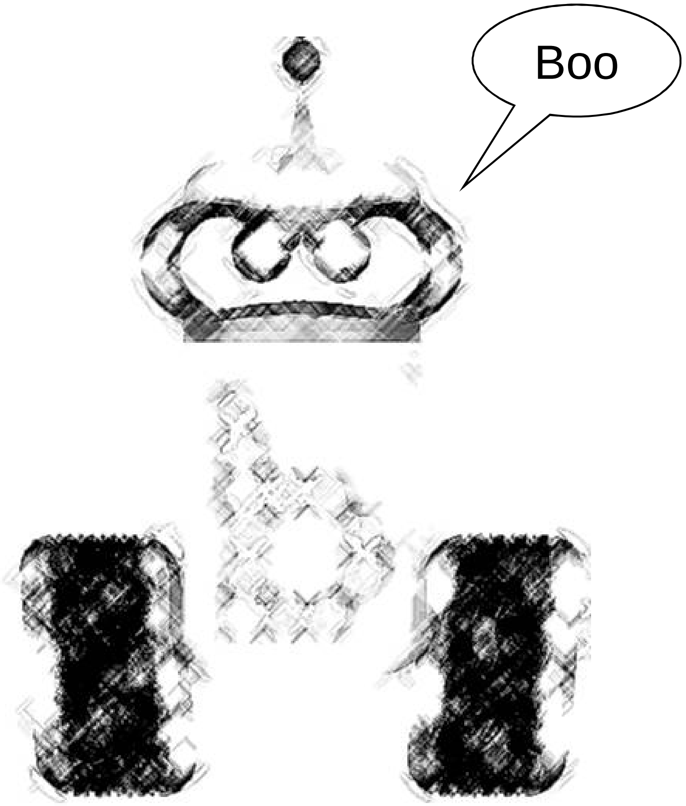
Science, technology, engineering, math, and computer science

Botball provides real life learning opportunities

Project management, teamwork, communication, and leadership

The Spirit of Botball

- Botball is an educational experience for students
- Parents, teachers and mentors are there to guide, **<u>not to *do*</u>**
- Adults who want to *do* should build practice boards and work on an entry for the KIPR Open
- Parents and mentors should set good examples of behavior and sportsmanship -- especially at tournaments



Successful Botball Team Members . . .

- Communicate and work well as individuals and as a team
- Understand how to look at a challenging situation and break it into solvable problems
- Think about how to apply what they've learned to other things outside of Botball
- Try a lot of approaches just to see what happens
- Have fun

For a Successful Botball Entry

- **Organize** your team: people, time, equipment
- **Think** about the problem before building
- Pick parts of the problem to **solve**
- Build a team of robots that **complement** each other
- Build software and hardware **together**
- **Document** everything and use it for team communications
- Spend as much or more time **testing** and tuning as building
- Use **checklists**
- **Observe** what other teams do at regionals and then use what you **learn** to make an improved entry for GCER

Thank you for participating!

We couldn't do it without you!

Botball®

Activity 0

The KISS IDE

(KIPR Instructional Software System
Integrated Development Environment)

Objectives

The KISS IDE

- Update the Firmware on your KIPR Link
- Install the KISS IDE on your personal computer

Charging the KIPR Link Controller

recap

- For charging the KIPR Link, **use only the power supply which came with your Link**
  - **Damage to the Link from using the wrong charger is easily detected and will void your warranty!**
- The KIPR Link power pack is a lithium polymer battery so the rules for charging a lithium battery for any electronic device apply
  - You should NOT leave the unit unattended while charging
  - Charge away from any flammable materials and in a cool, open area

KIPR Link Firmware Update Procedure (needed only if firmware isn't up to date)

1. Slide the power switch to turn on the KIPR Link: NOTE – the screen will turn off for several seconds when booting
2. Home screen has an *About* tab to identify the firmware version
3. To update firmware first connect the KIPR Link Charger to your Link
4. Copy the file *kovan_update.img.gz* from the current firmware image folder to a USB memory stick (version number is on folder name)
5. While holding down the side button on the KIPR Link (opposite side from the power switch) turn on the Link
   - The normal splash screen will first come up, followed by a screen instructing you to insert the flash drive
6. Insert the flash drive
   - A progress bar will appear and in a short time the update will complete
   - Your Link will then come up with its opening screen
   - **Click on *Settings* >> *Calibrate* to calibrate screen touch settings**

The KISS IDE

(Integrated Development Environment)

- The KISS IDE is "donation ware"
  - It is free and can be freely distributed and used for personal and educational purposes
  - If you like it and are looking for a tax deduction, please make a donation to KIPR
  - If you would like to use the KISS IDE in a commercial product, contact KIPR about licensing
- The latest version for the KIPR Link will be found at:

  http://www.kipr.org/products/kisside

  - It is also included in the electronic media distributed to participants of Botball workshops through the Team Home Base

The KISS IDE Installation

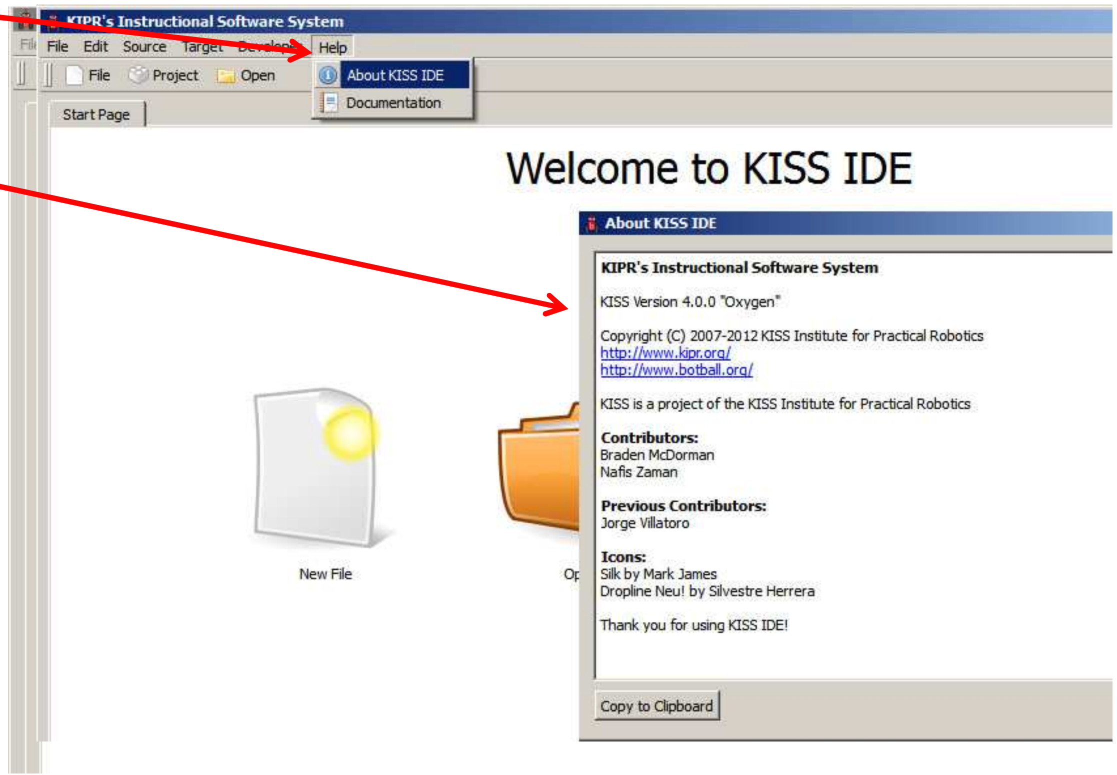
- Installation is the same as for most other software applications (the Team Home Base download site also includes instructions)
- The IDE supports Mac OS versions from 10.6 up, and Windows from XP through 7
  - For Windows, using the KISS IDE with the KIPR Link requires installation of the USB driver software that is included with the KISS IDE installer
    - If you are running Windows 7 you will probably need to right click on the installer .exe file so it runs in administrator mode
  - If you are using a Mac you will need to install the Developer Command Line Tools on your system (unless you did so at some earlier time)
    - For OS versions from 10.7 and up, the tools are a 171Mb download from either the Apple Developer or KIPR web site. For version 10.6 you will need to install Xcode 3 to get the command line tools. There will be media at the workshop for the different OS versions. Note: Command Line Tools are not installed with Xcode 4.

Using the KISS IDE

- The KISS IDE includes facilities that simplify the production of programs for the KIPR Link robotics controller
- The KISS IDE editor and robot simulator can be used whether or not a robot controller is connected
- Programs can be checked for syntax errors such as typos from within the KISS IDE interface
- To check for logic errors you:
  - Can simulate execution of your program using the KISS IDE's built in graphical simulator or
  - Attach a KIPR Link controller and try running your program

KISS IDE Version

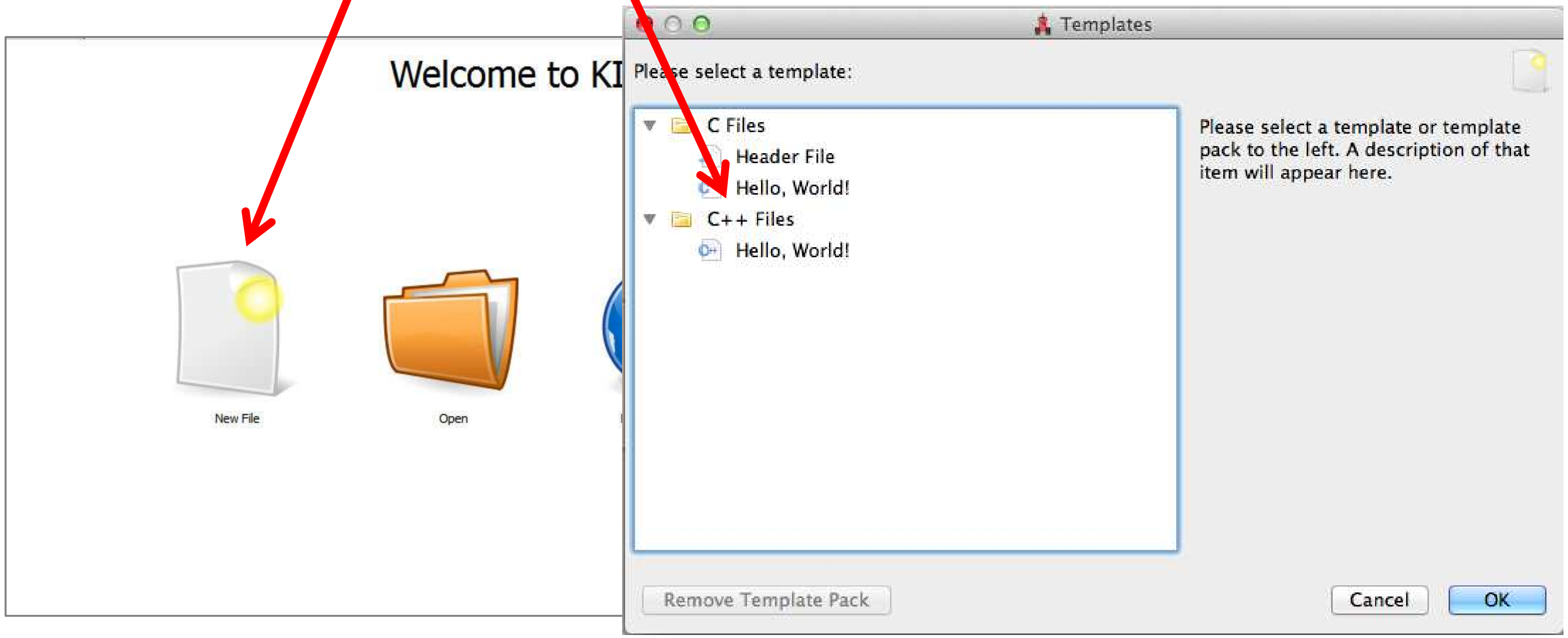
- If your installation was successful, a KISS IDE icon should now be on your computer’s desktop (Win) or in your Applications folder (Mac)
- Start the KISS IDE by clicking on the icon; the Welcome screen will appear
- Click on Help and select "About KISS IDE"
- Verify version number is current
  - Version 4.0 or higher
  - For the latest version, check **http://www.kipr.org/products/kiss-platform**
  - If you don't have the current version, download and install the new one



Launch the KISS IDE

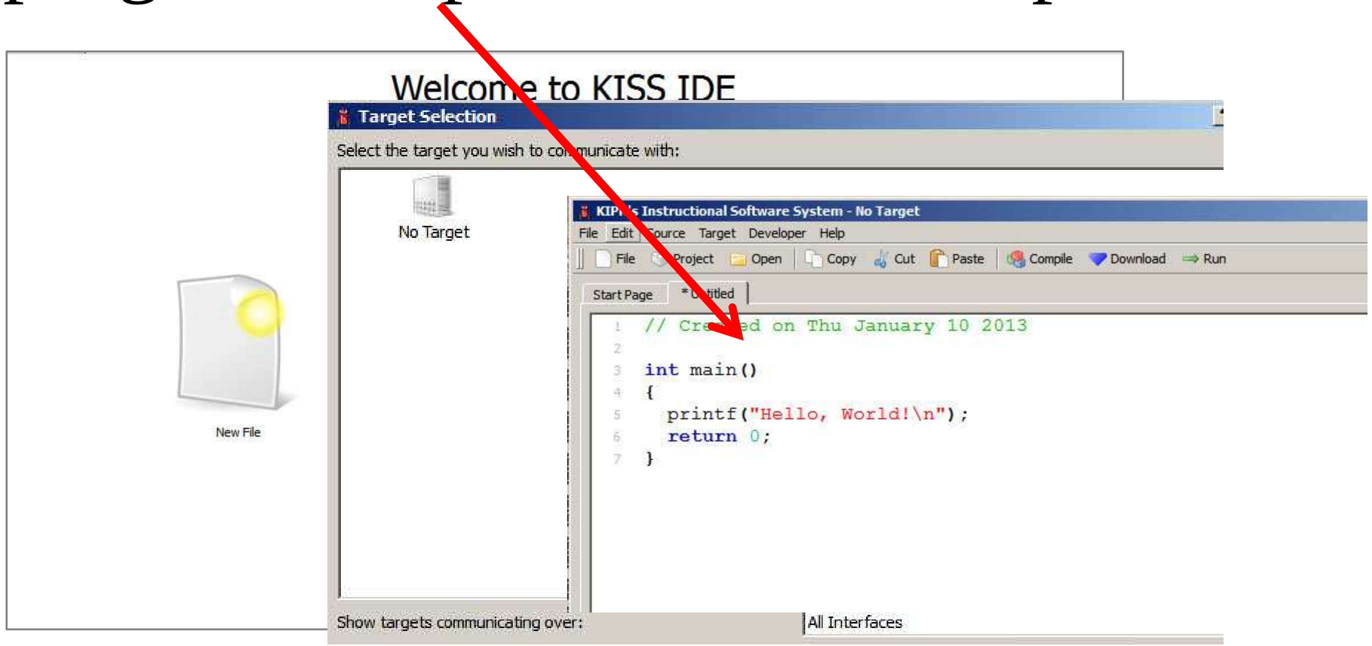
- Start the KISS IDE by clicking on its icon to get the Welcome screen
- Click on the *New File* icon and

  and choose the *Hello, World!* Template

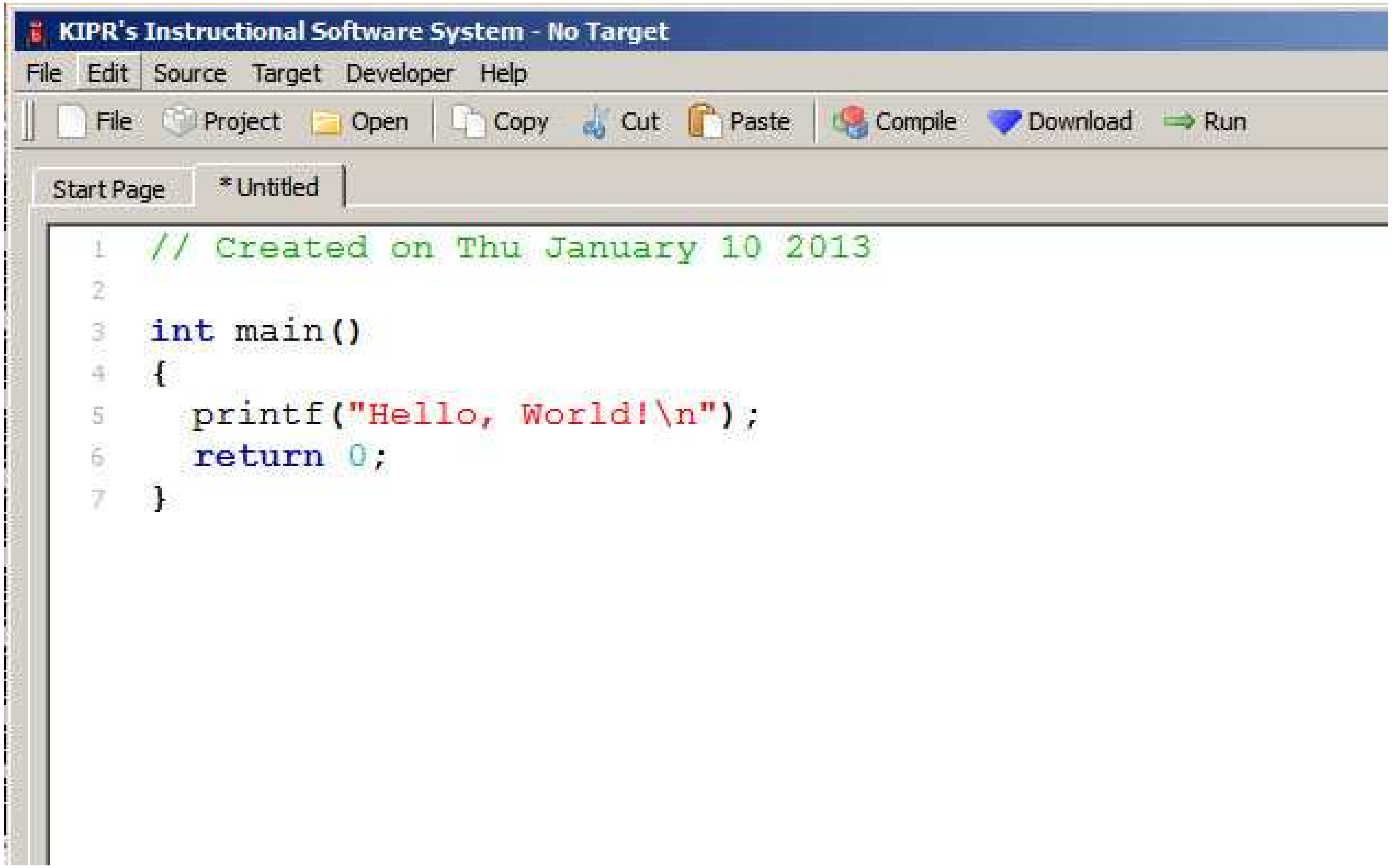


Select Target

- A Target Selection window will appear
- Pick *No Target* for now and the **C** program template will come up



The C Template: Hello, World!



Why **C**?

- **C** is a high level programming language developed to support the Unix operating system
  - The KIPR Link controller utilizes a version of Unix called Linux
- **C** is the most widely used language for systems programming
- Botball robots need to be programmed at the systems level to take advantage of the features of the KIPR Link
- For Botball, the KISS IDE (Integrated Development Environment) provides a user friendly interface to develop Botball programs in **C, C++ (Java and Python – soon)**
- For this workshop, we will focus on **C**

First C Program

Objectives

First **C** program

- Write a simple **C** program and run it on the simulator included within the KISS IDE

Prep

First **C** program

- Demo of a very simple program
  - Discussion of what is a **C** program
  - Where to find KISS IDE help
  - New file template
  - Running the program and observing results
- Detailed explanation of the simple program
- Recap of the step by step process

Simple Program Demo

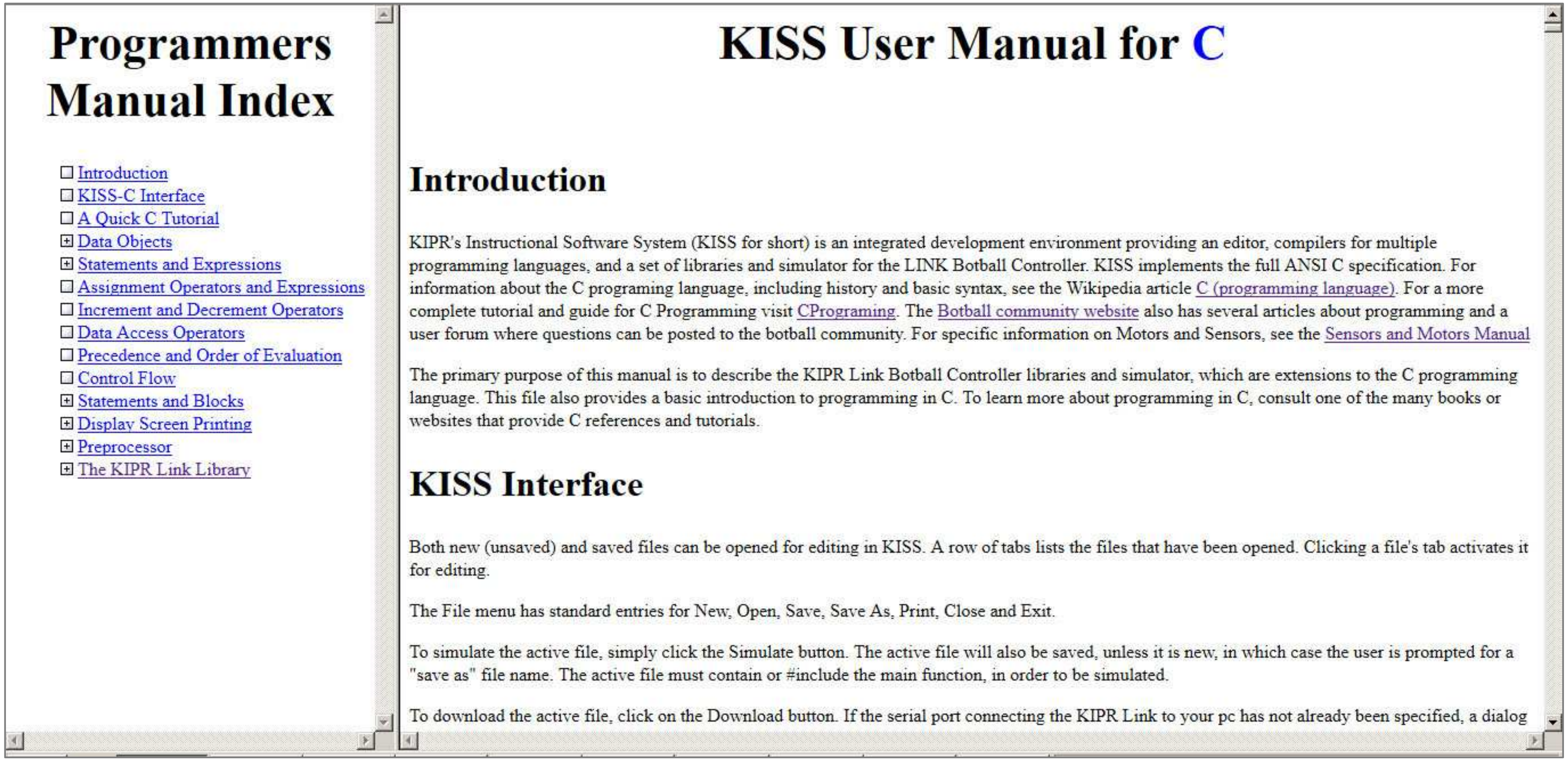
- We'll enter a simple program now
  - Your target selection needs to be *Simulator*
- When you click on *Run*, the KISS IDE **saves**, **compiles**, and **executes** your program on the simulator
  - *Compile* is the process of converting your program from ordinary text into a format the selected target understands
  - *Run* causes the target to execute your program
  - Note that when you have created a new file, KISS IDE will ask for a file name. A C file name can contain text ending with a single period followed by the letter c
    - *Good:* My File.c
    - *Bad:* My.File.c

Where Can I Get Help?

- The KISS IDE has an extensive Help ***Manual*** including a brief **C** tutorial
  - A manual relevant to the programming language for the template selected can be found under the *Help* menu
  - When using **C** for Botball, the help file ***Manual*** is the primary document to consult
  - The manual covers the library of functions for accessing the features of the Link controller and for controlling a Create module
- The file ***Sensors and Motors Manual*** provides additional information about the sensors and motors used with the KIPR Link

KISS IDE Help Manual

- Accessed from the KISS IDE help tab



Programmers Manual Index

KISS User Manual for C

Introduction

KIPR's Instructional Software System (KISS for short) is an integrated development environment providing an editor, compilers for multiple programming languages, and a set of libraries and simulator for the LINK Botball Controller. KISS implements the full ANSI C specification. For information about the C programing language, including history and basic syntax, see the Wikipedia article C (programming language). For a more complete tutorial and guide for C Programming visit CPrograming. The Botball community website also has several articles about programming and a user forum where questions can be posted to the botball community. For specific information on Motors and Sensors, see the Sensors and Motors Manual

The primary purpose of this manual is to describe the KIPR Link Botball Controller libraries and simulator, which are extensions to the C programming language. This file also provides a basic introduction to programming in C. To learn more about programming in C, consult one of the many books or websites that provide C references and tutorials.

KISS Interface

Both new (unsaved) and saved files can be opened for editing in KISS. A row of tabs lists the files that have been opened. Clicking a file's tab activates it for editing.

The File menu has standard entries for New, Open, Save, Save As, Print, Close and Exit.

To simulate the active file, simply click the Simulate button. The active file will also be saved, unless it is new, in which case the user is prompted for a "save as" file name. The active file must contain or #include the main function, in order to be simulated.

To download the active file, click on the Download button. If the serial port connecting the KIPR Link to your pc has not already been specified, a dialog

Templates and Comments

- For starting a new **C** file for the KIPR Link you should use the *C* template, or one of your own creation (covered later)
- Two ways to comment **C** programs

```
// is a comment for rest of line
```

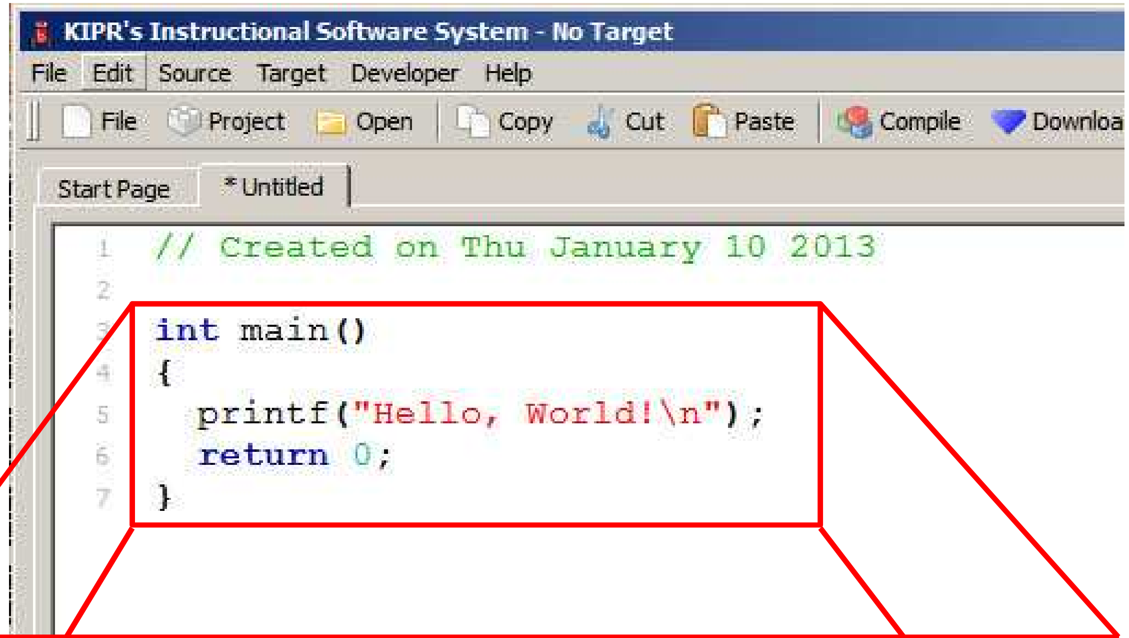
or

```
/* is a comment that goes from
   the initial slash-star till
   the first star-slash */
```

A Simple C Program

By default, the KISS IDE colors your code and adds line numbers

- Comments appear in green
- Key words appear in **bold blue**
- Text strings appear in red
- Numerical constants appear in aqua
- Compiler directives appear in blue



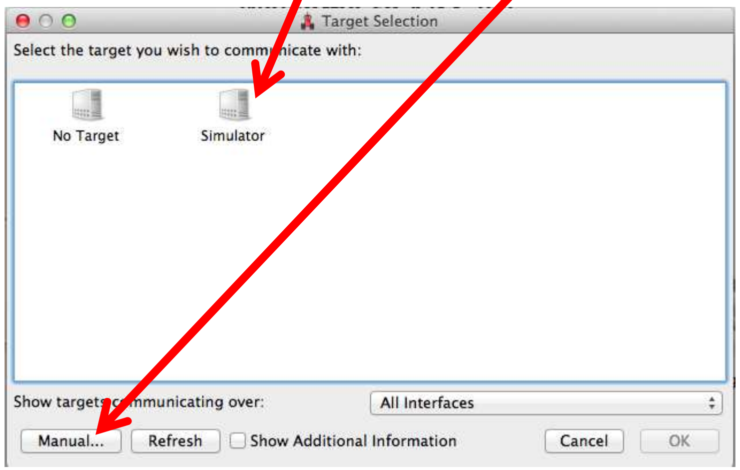

```
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

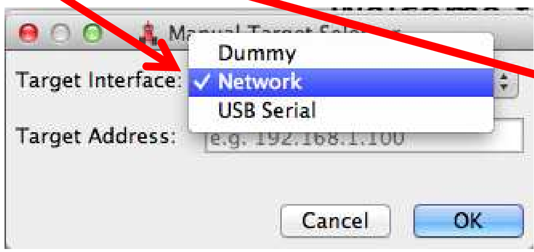
Select Simulator Target

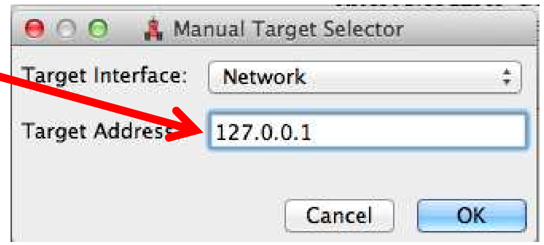
- Start the ks2 (simulator) application
- Click *Target >> Change Target*
- Choose *Simulator*

OR

- If the simulator does not appear as a target, verify that the ks2 application is running and
    1. Click on the *Manual* button
    2. Select *Network* from the drop down menu
    3. Type in the address *127.0.0.1* and click *OK*

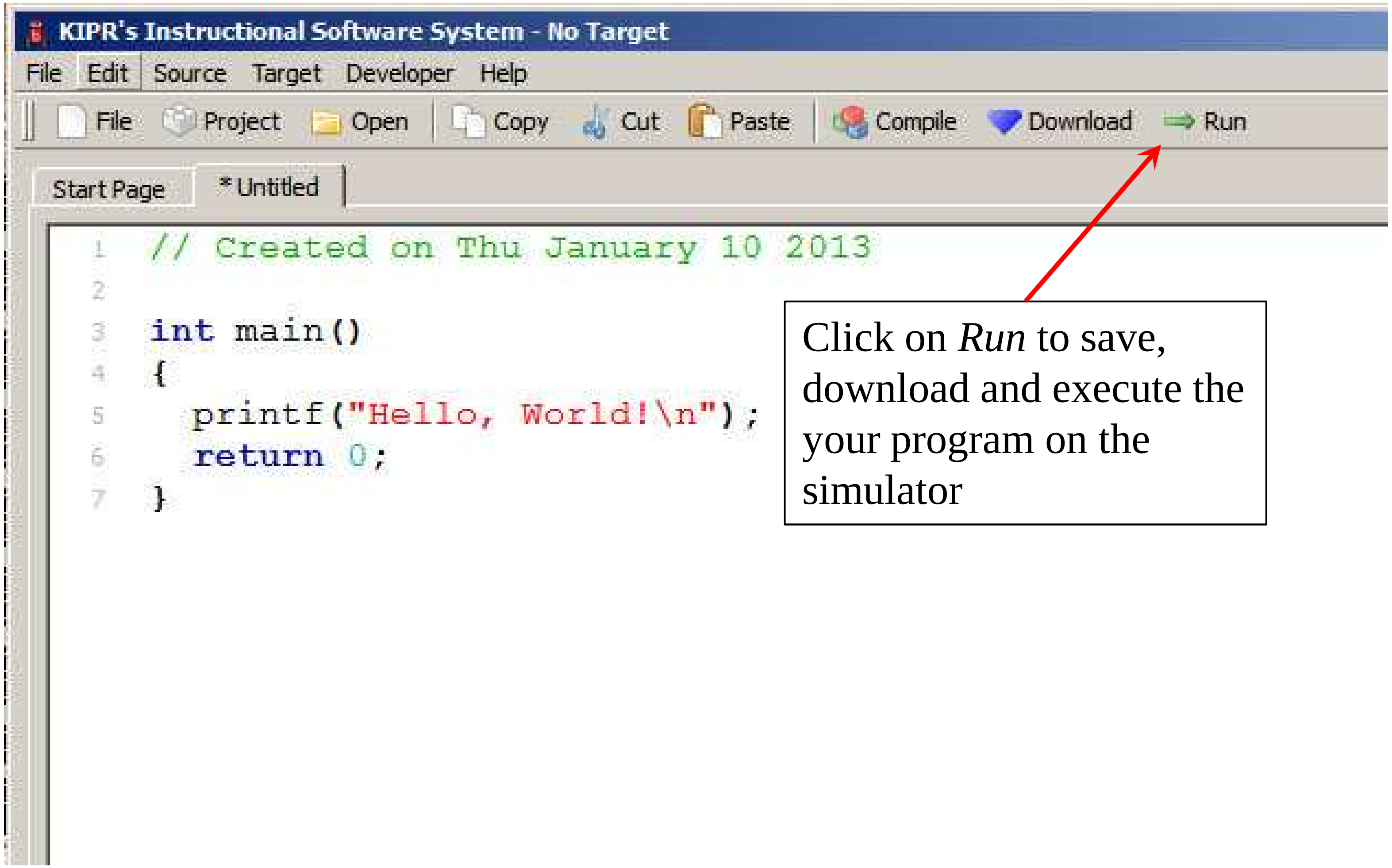






Botball®

Executing the Program in the Simulator



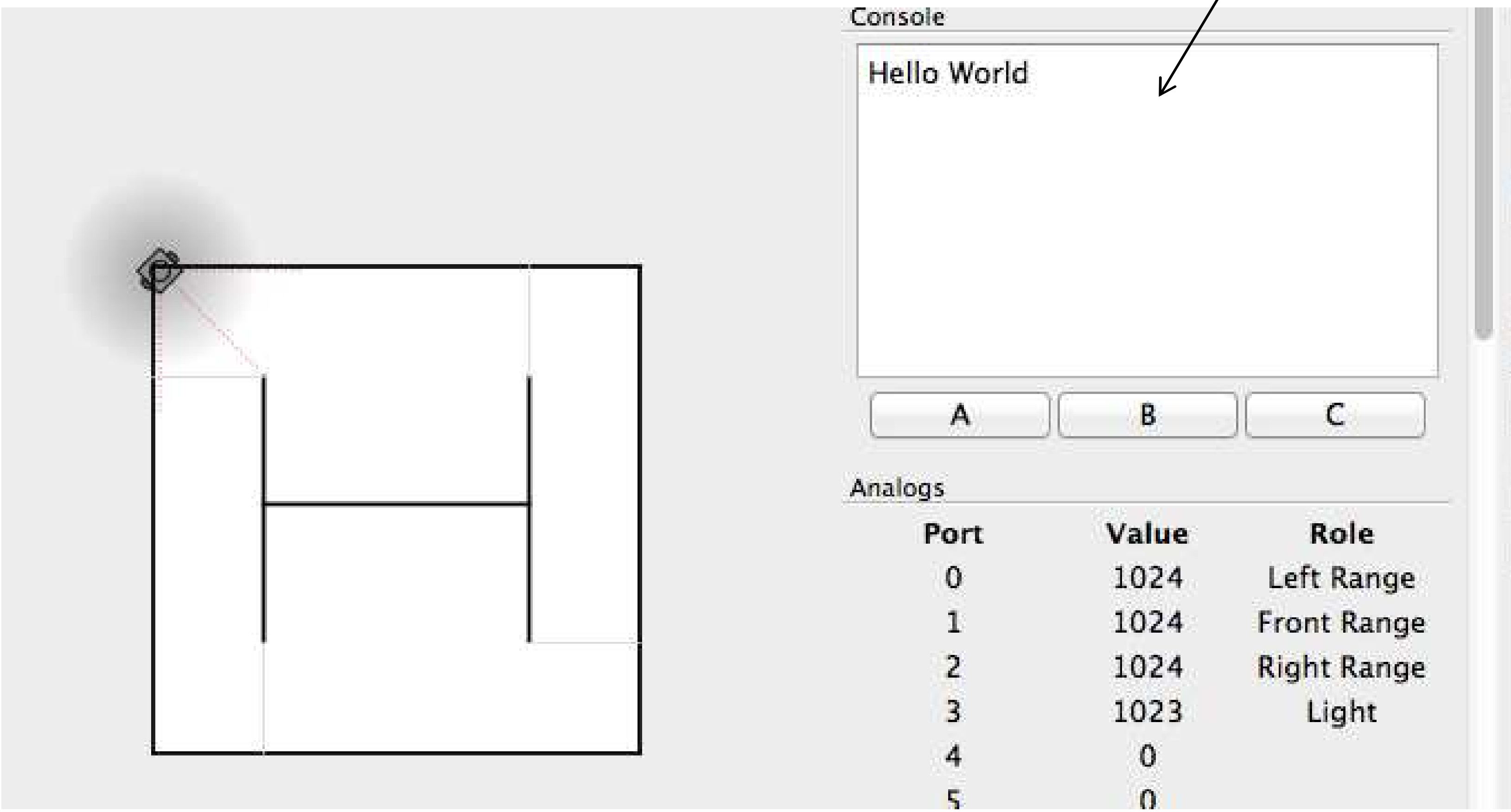
Click on *Run* to save, download and execute the your program on the simulator

Using the Simulator

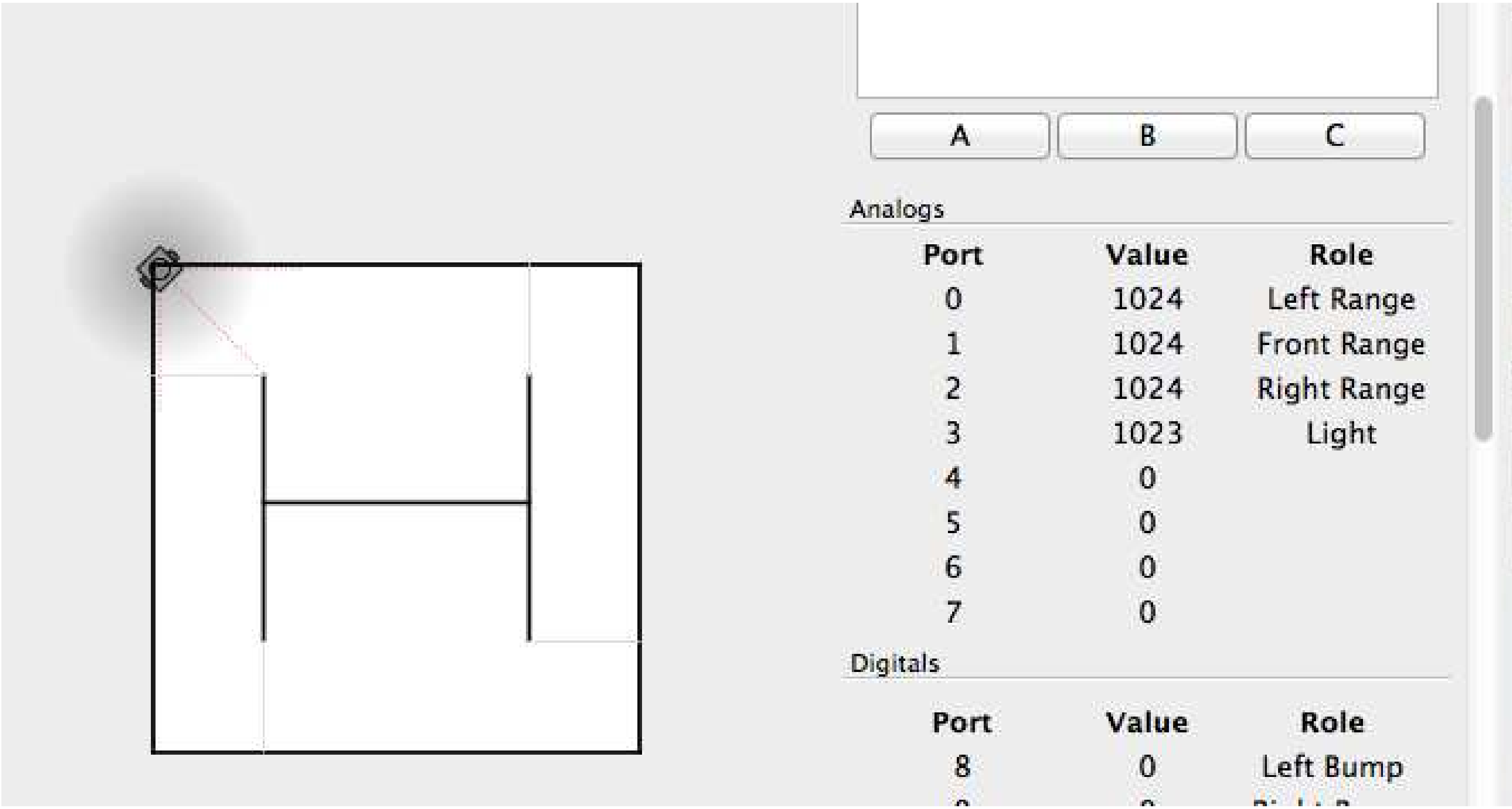
- For programs that have their target set to the simulator:
  - Click **Run** in KISS to save download, compile and run your program
  - Clicking on the simulator window prior to pressing run allows the robot and light to be placed.
  - The light can be switched on and off by double clicking
- The simulator pre defines several sensors and motors and where they are connected to the simulated KIPR Link
  - The robots you build may have different arrangements.

Observing Results

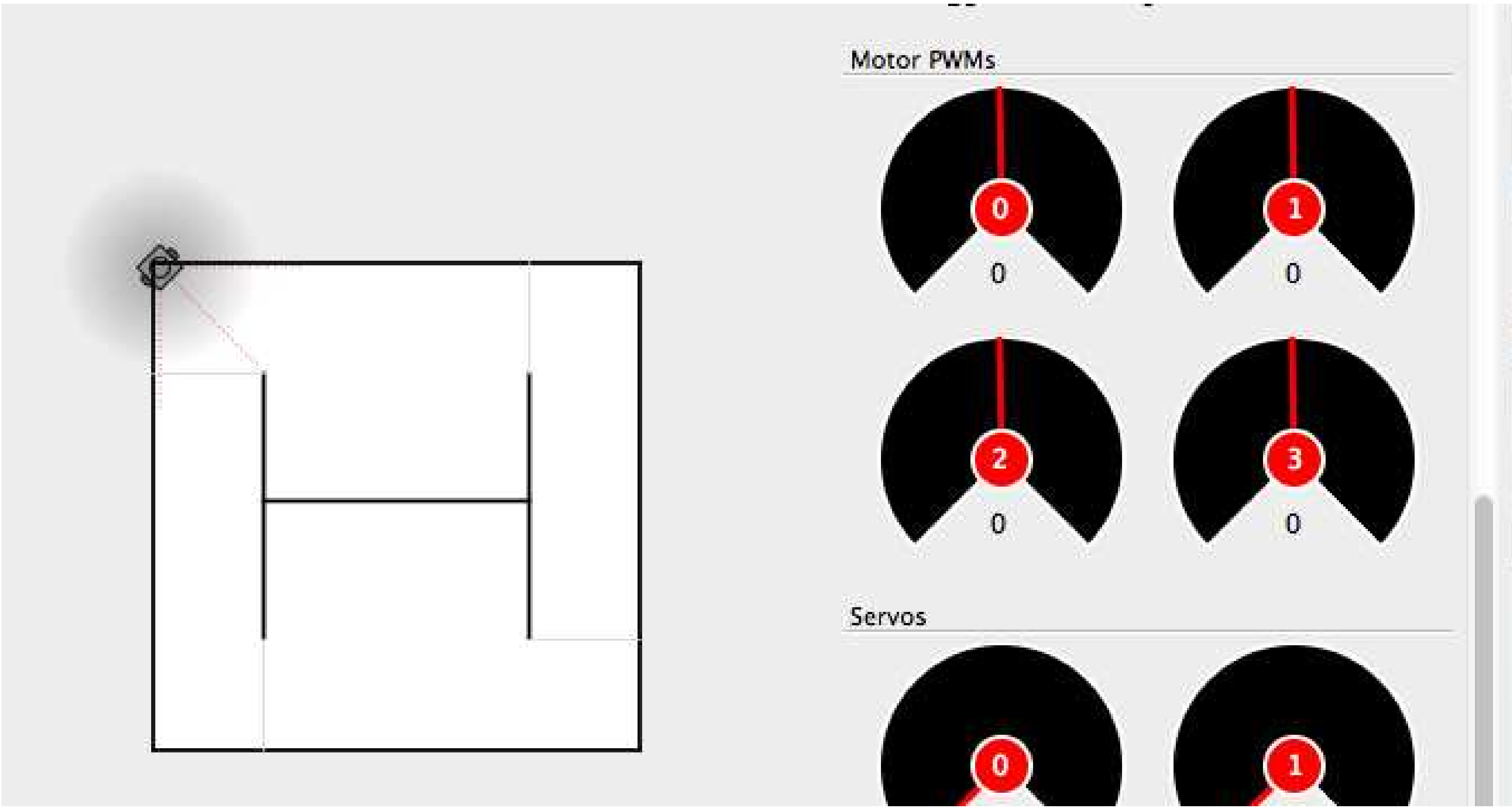
Your program's output



Simulator's Scrolling RHS Pane



Simulator's Scrolling RHS Pane



The Program Explained

(it illustrates most **C** syntax)

Function Definition

return type *name* *argument list*

```
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Blocks of Code

```
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Function Calls

```
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Terminating Statements

```
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Functions Return a Value
(even **main**)

```
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

msleep()

- Like **printf()**, **msleep()** is a built-in (library) function
- **msleep(3000)** causes the KIPR Link to pause for 3 seconds
  – Example :

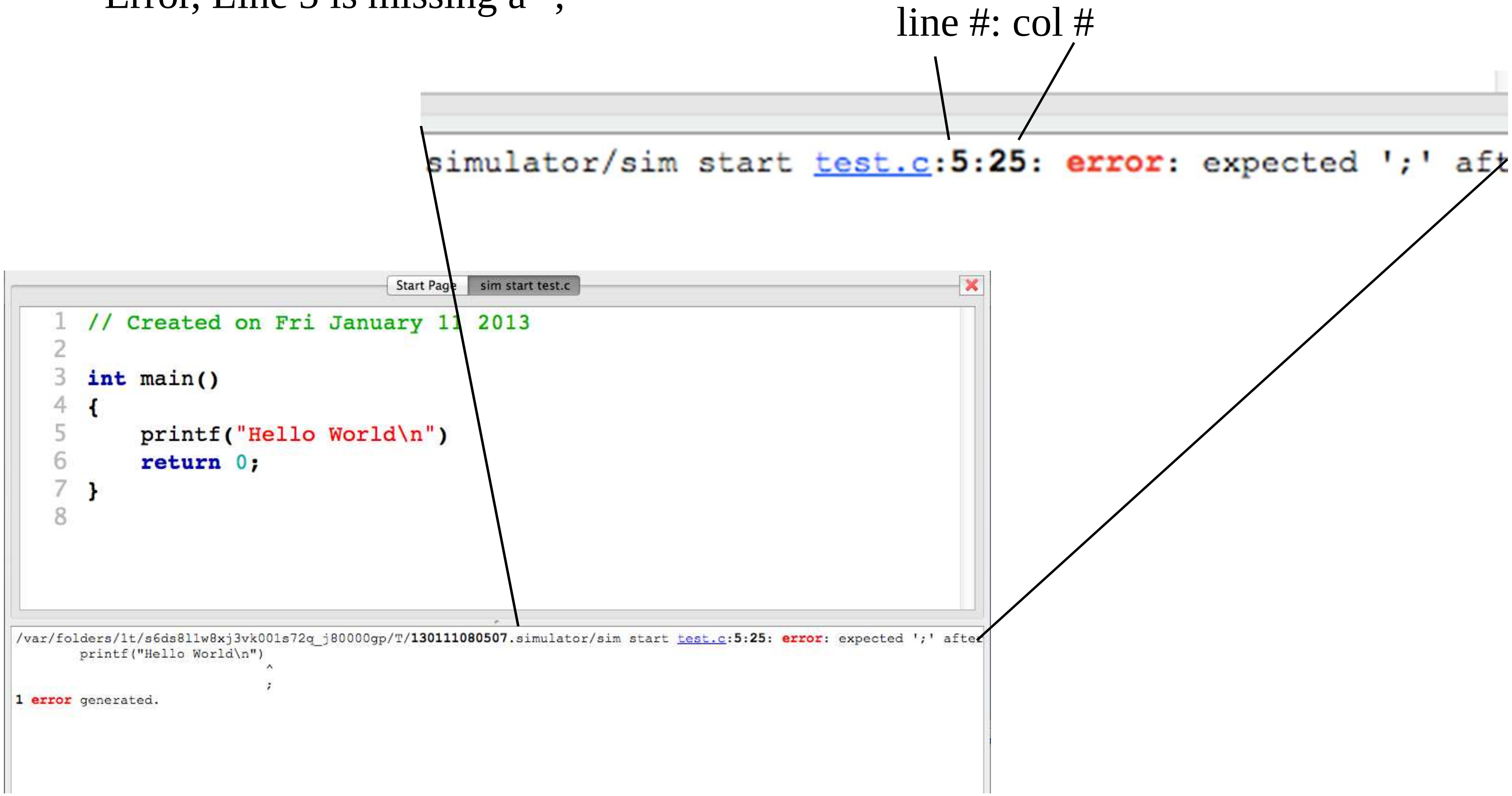
```
printf("slow  ");
msleep(3000);
printf("reader\n");
```

Step by Step Process

- Start the IDE and click on the *New File* icon
- Select *Basic C File* Template and the *Simulator* communications target
- Modify the **C** program template to create your own program
- At any time you can save the file by using *File*..*Save* or *File*..*Save As*
  - File names can contain: letters, numbers, – and _
  - Unless you override it, the KISS IDE will append .c to the file name
    - Required to compile and run the program
    - Except for extension, do not use a "." in the file name
  - The directory where you last saved becomes the default directory for saving files until you change to another directory
- Check your program by clicking on the *Run* button
  - This automatically saves your program, then runs it if there is no error
- If there is an error, the KISS IDE will give a message and a line number, marking the line with a red dot (fix the first error, then recompile!)
  - **the error will be on OR <u>before</u> that line**

Example Error Message

Error, Line 5 is missing a ";"



Activity 1 (Objectives)

Programming Basics and Screen Output

Write a program for the KIPR Link that displays "Hello World!" to the screen, delays two seconds, and then displays your name on the screen.

Execute the program on the simulator.

Activity 1 (Task Design)

Programming Basics and Screen Output

Break the objectives down into separate tasks and think about how each might be accomplished; for example, the larger task might be developing a program to operate a robot's claw, which has tasks within for making the claw open or close. Since this is our first example, the tasks are pretty simple:

1. Display "Hello World!" on the screen.
2. Delay for 2 seconds.
3. Display your name on the screen.

Activity 1 (Program)

Programming Basics and Screen Output

Use our previous Task Design as *Pseudocode* (this means "false code") to help write the real code…

1. Display "Hello World!" on the screen.
   – Use **`printf()`** function
2. Delay for 2 seconds.
   – Use **`msleep()`** function
3. Display your name on the screen.
   - Use **`printf()`** function

Comment your code (pseudocode makes great comments) - your comments show what you think you told your bot to do, but not necessarily what it will actually do!

Activity 1 (Experiments)

Programming Basics and Screen Output

- Try adding more **`printf()`** statements to your program (pay close attention to the syntax, particularly the terminating semi-colon needed by each statement)
- Have the program print out a haiku about robotics
- Execute your revised program (*Run* button)
- Experiment by leaving off or adding extra "**`\n`**" or "**`\t`**" to the start or end of the strings in your **`printf()`**
- Try adding the command **`display_clear();`**
- Can you print out more lines than can show on the screen at one time?
- Have the program print out a poem or Haiku. What happens when the screen fills up?

Activity 1 (Solution)

Programming Basics and Screen Output

```
/**************************************************************
*********  Activity 1
**************************************************************/
int main()
{
       // 1. Display "Hello World!" to the screen
       printf("Hello World!\n");

       // 2. Delay for 2 seconds
       msleep(2000);    //2000ms = 2s

       // 3. Display your name to the screen
       printf("My name is Botguy.\n");

       return 0;
}
```

Activity 1 (Reflections)

Programming Basics and Screen Output

- What have you learned from this activity?
  - e.g., what did you find out from the simulator
  - or what did you figure out in messing with the simulator?
- What does the "**`\n`**" and "**`\t`**" do in **`printf()`**?
- What does **`display_clear()`** do?
- How many lines can you see on the screen at once? What happens to the others?

Operate Simbot

`while` statement

- Programs normally move from one statement to the next
- Sometimes we have a block of statements we wish to repeat until some event takes place
- A **`while`** statement is used to accomplish this task by checking to see if something is true
  - Tests that check if something is true or false are called Boolean operations
  - More information on "Boolean operators" (such as **`==`**, **`<=`**, **`>=`**, **`!=`**, **`<`**, etc.) is in the KISS IDE help
    - **`==`** (two equals signs together, not one) is used to test if two values are the same

Background

Robot Simulator – Driving

1. The simulated robot has motors and wheels plugged into motor ports 0 and 2
    1. the command motor(<port>,<power>) is used to control motors
    2. there are 4 possible motor ports, 0-3
    3. motor power levels can be anywhere from -100 to 100 with 0 being off. The function ao() turns all 4 motors off.
2. The simulator has several sensors including a left bumper in digital port 8 and a right bumper in port 9
    1. the function digital(<port>) returns 0 if that bumper is not pressed and 1 if it is pressed
3. There are light sensors on analog ports 3 and 4
    1. The function analog10(<port>) returns the value of that sensor 0-1023
    2. Light sensors give small numbers for bright lights and large numbers for dark.

Function Examples

- motor(0,100);
  - Turns on motor 0 at 100% power
- digital(8);
  - returns 0 if the left bumper is not pressed and 1 if it is pressed
- analog10(3);
  - returns the value of the left light sensor, Light sensors give low values for bright lights and high values for dark

Pseudocode: Drive Till Bump

- Announce what the program does
- While the left AND right bumpers are not pressed (have value of 0)
  - Turn on the left motor
  - Turn on the right motor
- When a bump occurs turn off the motors
- Announce that the program is done

Program

Simulator – Driving forward

```
/*****************************************************************
Drive the simulated robot forward at half power till it bumps
*****************************************************************
*/
int main()
{
  printf("Drive Straight till bump\n"); // announce the program
  msleep(1000); // wait 1 second

  while (digital(9)==0 && digital(8)==0) // check bumpers
  {
     motor(0,55); // Drive left wheel forward at 55% power
     motor(2,50); // Drive right motor forward at half power
  }

  ao(); // Stop motors
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
```

Activity 2 (Objectives)

Programming Basics and Screen Output

Write a program for the KIPR Link that waits for a light to turn on, then drives until the robot runs into a wall. You can go straight or in a curve

Execute the program on the simulator.

- Drag the robot to the desired starting position
- Drag the light near the robot
- Double click the light to turn it on

Activity 2 (Task Design)

Programming Basics and Screen Output

1. Announce the objectives.
2. Start when the light comes on.
3. Drive forward.
4. Stop when bumper pressed.
5. Announce program is over.

Activity 2 (Program)
Programming Basics and Screen Output

1. Announce the objectives.
   – Use **`printf()`** function
2. Start when the light comes on .
   – Use **`while`** to check **`analog10(3)>500`**
3. Drive straight.
   - Use **`motor()`** functions like previous example
4. Stop when bumper pressed.
   – Use **`while`** to check **`digital(9)`** and **`digital(8)`**
   – Use the **`ao()`**functions like previous example
5. Announce program is over.
   – Use **`printf()`** function

Botball®

Activity 2 (Experiments)

Programming Basics and Screen Output

- Try changing the motor speeds
- Try different values for the light sensor
- Try moving the light source farther away from your robot in the simulator
- What happens if you drive your robot backwards?
- What if you use:

```
while (digital(9)==0 || digital(8)==0)
```

Instead of `while (digital(9)==0 && digital(8)==0)`

Program

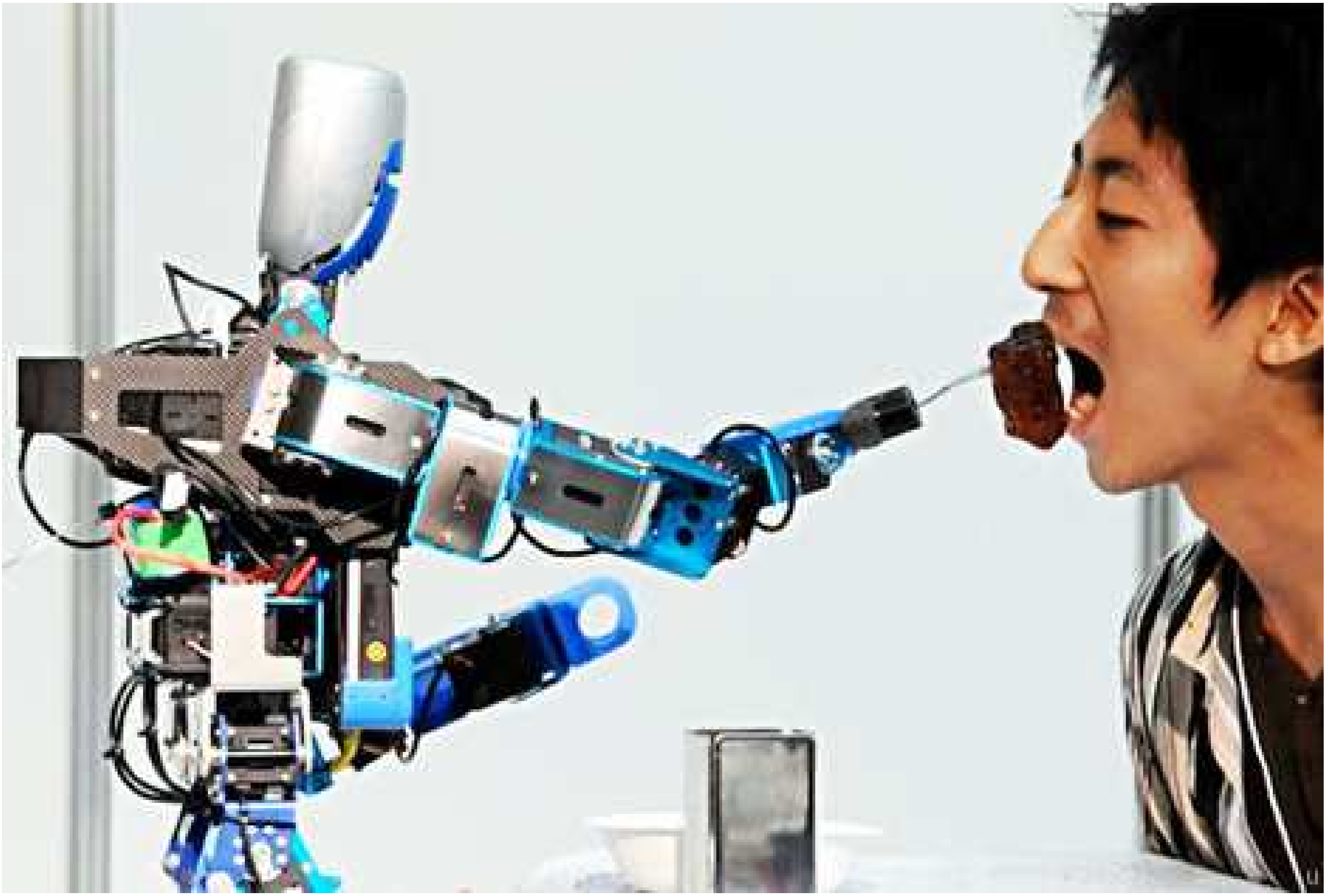
Simulator – Start with Light and Driving Straight

```
/*********************************************************************
After the Light comes on drive the simulated robot forward at half power till
it bumps
*********************************************************************/
int main()
{
  printf("Turn on light to Drive Straight until bump\n"); // announce

  while (analog10(3)>500) {}//while light value is above 500 do nothing
                            //go on when value is below or equal to 500

  while (digital(9)==0 && digital(8)==0) // check bumpers
  {
     motor(0,55); // Drive left wheel forward at 55% power
     motor(2,50); // Drive right motor forward at half power
  }

  ao(); // Stop motors
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
```

Activity 3: BUILD DemoBot

Use the DemoBot Building Guide

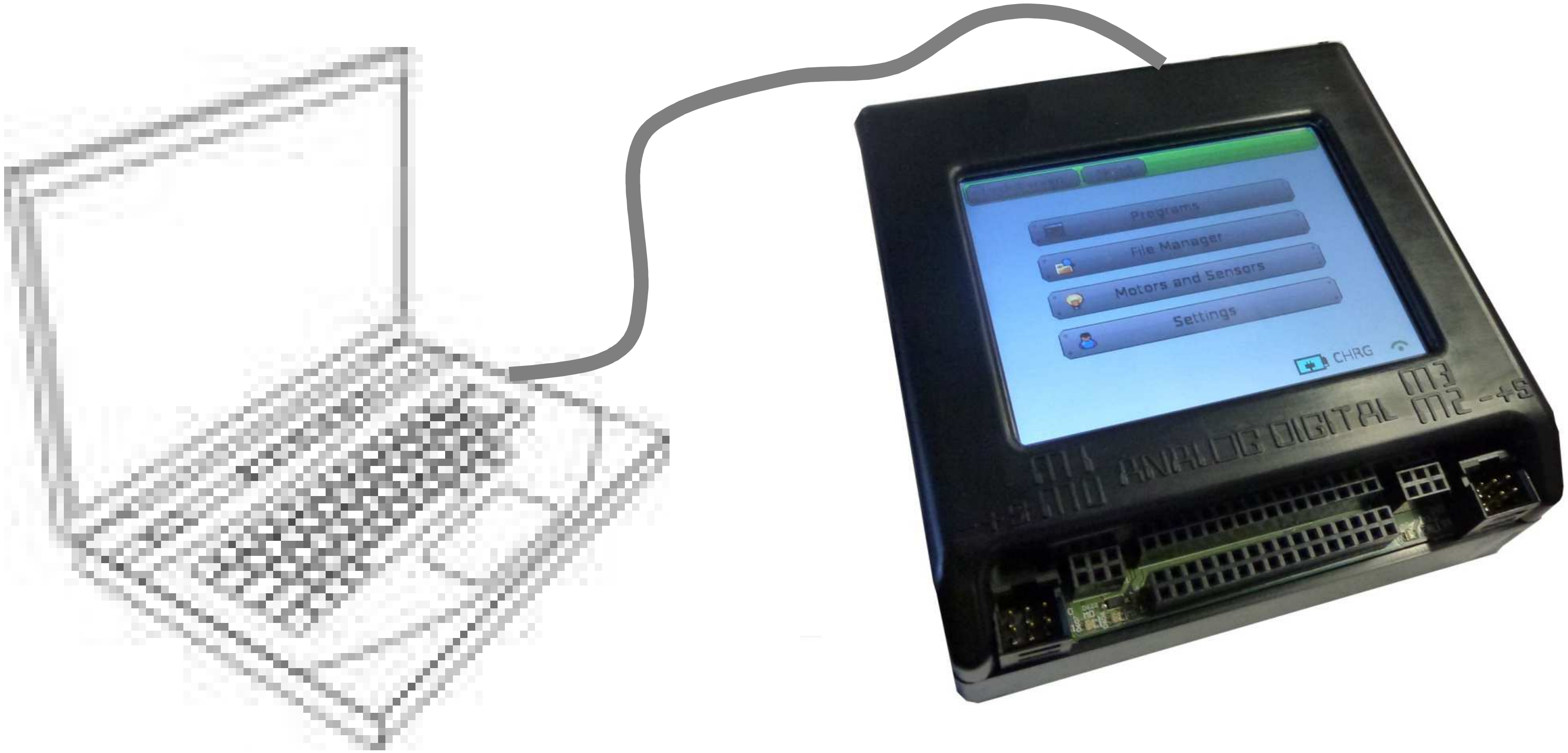
LUNCH



Operate a Create Module Using a KIPR Link

Attach Computer to KIPR Link

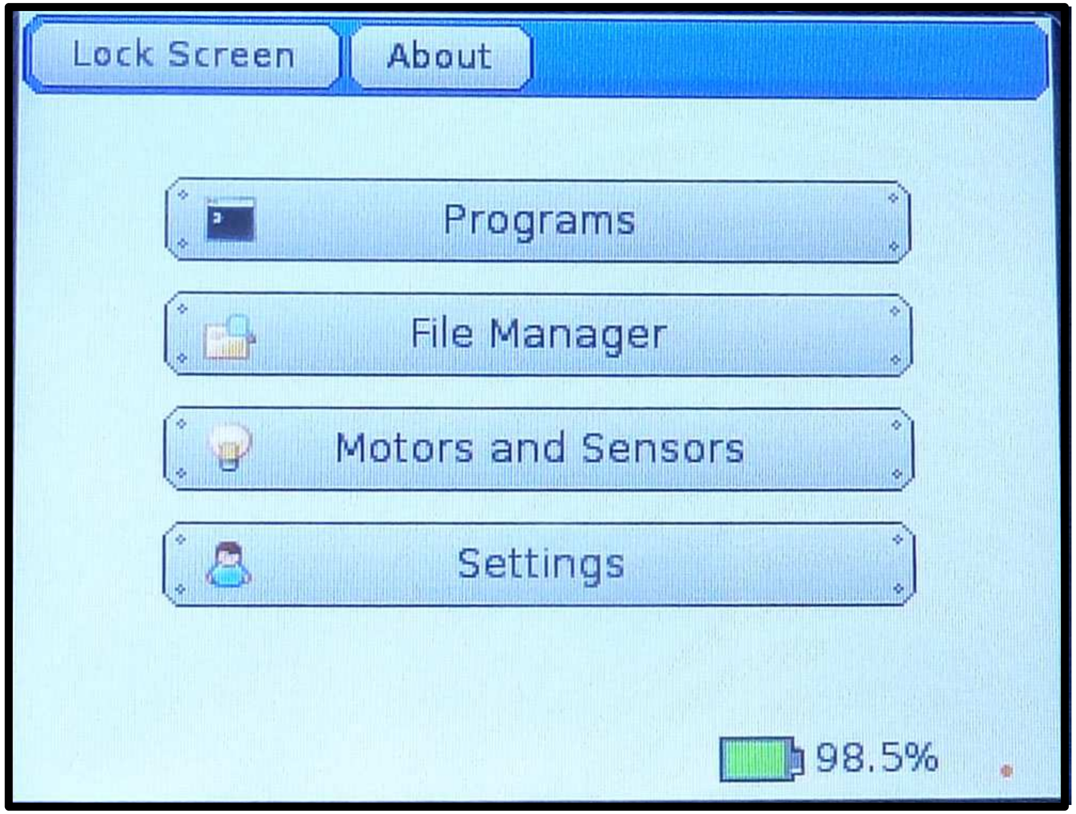
USB Cable in Botball Kit



See quick start guide on Team Home Base

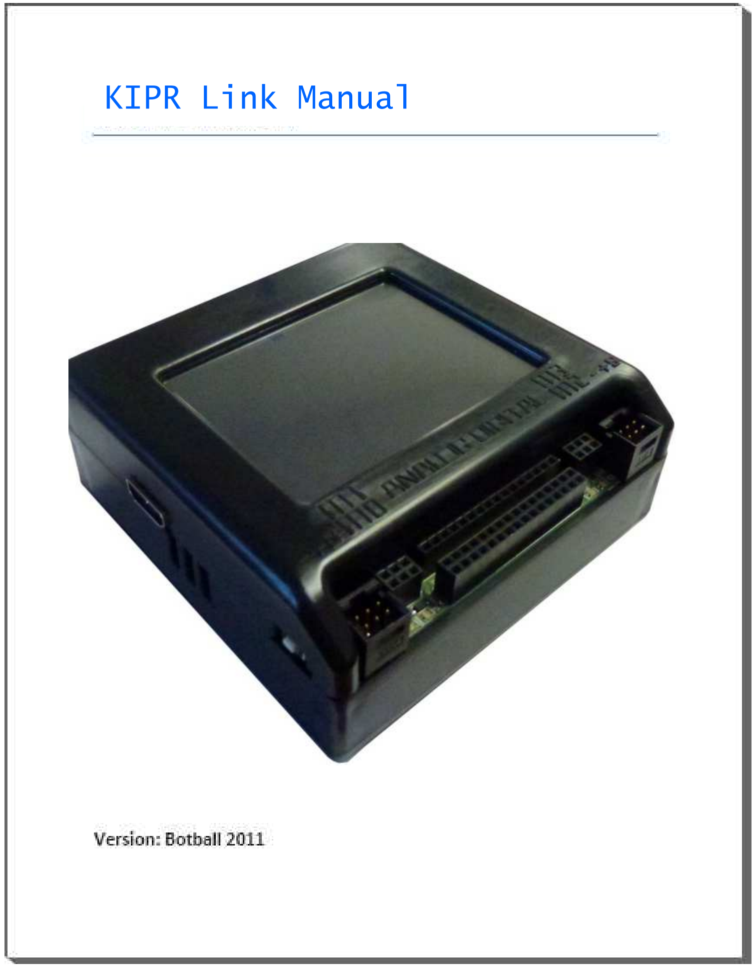
Power Up the KIPR Link

Opening Screen



KIPR Link Manual

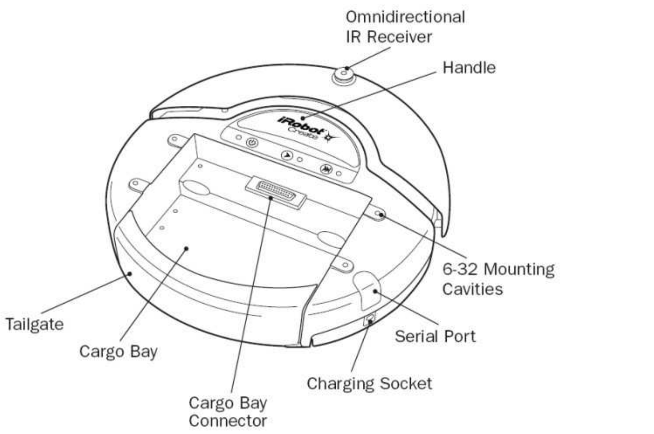
- For further detail about the KIPR Link, consult the KIPR Link Manual on your Team Home Base

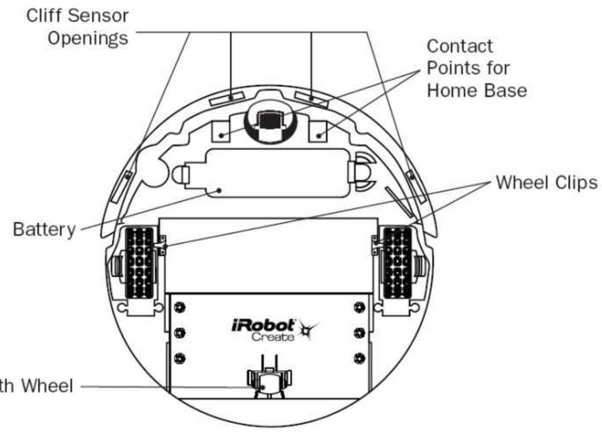


Running Your Program on the KIPR Link

- When you compile a program to the KIPR Link it is automatically downloaded and made ready to run
- Under the Programs Menu, select your program
- Pressing *Run* causes the specified program to run
- Your program code is retained on the KIPR Link in the Programs Menu until manually deleted

KIPR Link in Create Cargo Bay





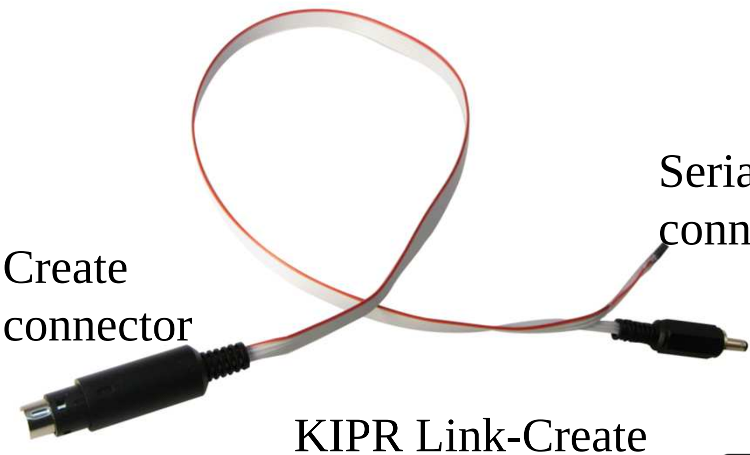
From iRobot Create specs

Install the **yellow** battery into the bottom of the Create – make sure it snaps in completely. Connect the Create charger when not using Create.

See Quick Start Guide for more details on connecting the Create to the KIPR Link

Connecting KIPR Link-Create Cable

- The KIPR Link-Create Cable is keyed, so it will only insert in the correct orientation
- The KIPR Link-Create Cable is keyed, so you will need to release the lock before removing – DON'T JUST PULL



KIPR Link-Create cable (in Botball kit)

- The Create will only TRICKLE charge the KIPR Link.

Moving the Create

- Objectives
  - commands for moving
  - commands for using Create sensors
  - combining commands to do something interesting

iRobot Create

Slide 90

CW35 does simulator do create?

Charles Winton, 1/10/2013

CW36

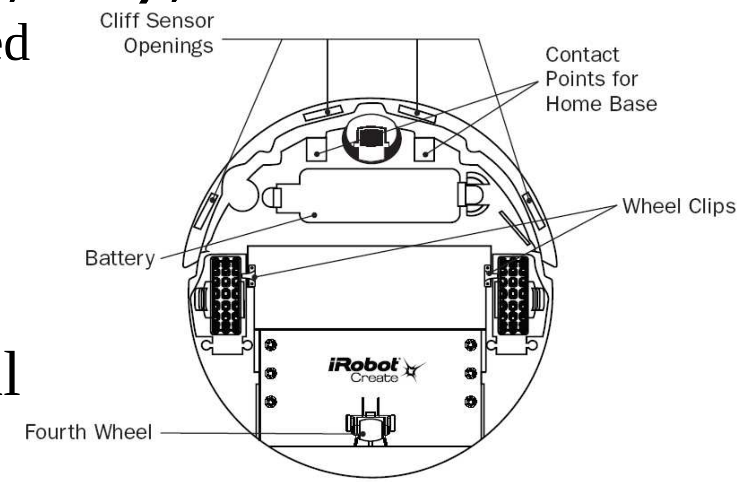
`create_connect()` & `create_disconnect()`

- The KIPR Link has a serial interface, which by default is set for downloading programs from the KISS IDE to your KIPR Link via a USB cable connection
  - Computer access to the Create module is also by serial interface
  - There is a cable in your kit to provide a connection between KIPR Link and Create
- The library function **`create_connect();`** sets the KIPR Link serial interface for the Create cable connection rather than the USB cable connection
  - The Create has to be turned on for this to work
  - Once connected, your KIPR Link can send commands to operate the Create
- The library function **`create_disconnect();`** sets the KIPR Link serial interface back to the USB cable connection and shuts off Create motors
  - Power cycling your KIPR Link also sets the serial interface to be for the USB cable connection (but it won't shut off Create motors!)

CW36 simulator reference
Charles Winton, 1/10/2013

Create Motor Functions

- **`create_drive_direct(`***left_speed***`, `***right_speed***`);`** specifies separate right and left speeds for the two drive motors – the command continues until a different motor command is received
  - Speed can range between -500 and 500 mm/sec
  - **`create_drive_direct(100,100);`** moves the Create forward at a modest speed
  - **`create_drive_direct(100,200);`** moves the Create counterclockwise at a modest speed
- **`create_stop();`** stops the drive motors
- There are more drive commands; see the Help Manual



Initializing Create Distance and Angle Calculations

- As the Create operates, the angle turned through and distance traveled are accumulated
- The functions **`set_create_distance(`***<val>***`)`** and **`set_create_total_angle(`***<val>***`)`** reset the distance accumulation to start from *<val>*
  - **`set_create_distance(0);`** initializes the distance accumulation to start at 0
  - **`set_create_total_angle(0);`** initializes the angle accumulation to start at 0

Create Status Monitoring

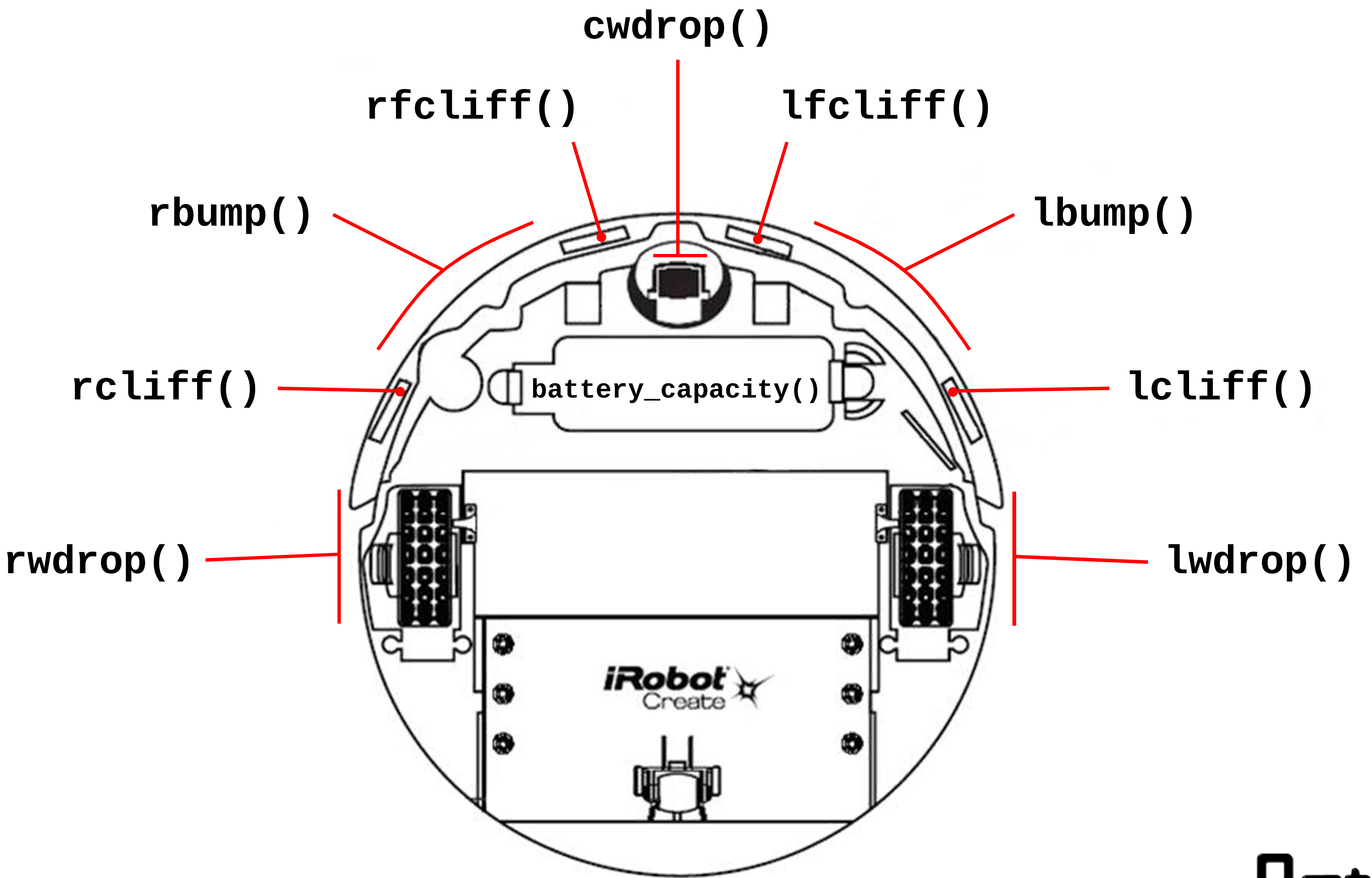
- Internal Create sensor data can be accessed by the KIPR Link using "**`get_create`**" library functions
  - **`get_create_distance()`** returns the distance (in mm) that the center of the Create has traveled
  - **`get_create_total_angle()`** keeps track of the number of degrees the Create has turned through
  - **`get_create_lbump()`** returns the value of the left bump sensor (pressed = 1, not pressed = 0)
- Other Create sensors have functions as well…

Slide 94

CW37 no lag needed?

Charles Winton, 1/10/2013

get_create_…


cwdrop()
rfcliff()
lfcliff()
rbump()
lbump()
rcliff()
battery_capacity()
lcliff()
rwdrop()
lwdrop()
iRobot
Create

External vs. Internal Sensors

- Internal sensors measure things going on inside the robot and are used to infer how the robot is moving
  - e.g., get_create_distance does not measure distance but instead measures how far the wheels have turned and assumes that robot moves accordingly
  - This is called dead reckoning
- External sensing like bump and light sensors measure how the environment interacts with the robot
  - If there is nothing between your robot and the wall, then when the bumper is pressed, you have probably reached the wall
- Combining the two types increases reliability
  - e.g., there is a wall 3m in front of you, if your program moves your forward and stops when the bumper is pressed, and if your distance sensor says you have moved more than 2 and less than 4m, then you are probably at the wall you intended.

Programming for Humans

- Programs should always announce their intentions:
  - Use **`printf`** and **`msleep`** to have the program print to the screen what the program does
- Whenever operator input is needed – make sure that the program prints out a prompt to the user
- Programs should announce when they are done

Demo

Driving Straight

Drive the Create forward until one of the bumpers is pressed

Slide 98

CW38 simulator
Charles Winton, 1/10/2013

Demo Code

```
int main()
{

  printf("Trying to connect to Create...\n");
  create_connect(); // Program stops till it connects...
  printf("Connected. Drive straight until bumper hits\n");
  while (get_create_lbump() == 0 && get_create_rbump() == 0 )
  {
    create_drive_direct(150,150); // drive straight at 150mm/s
  }
  create_disconnect(); // stops communication
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
```

Conditions and Functions

Activity 4 (Prep)

Conditions and Functions

- buttons
- `if-else` statements (and `else if`)
- functions

Buttons

- There is 1 physical (named *side*) and 6 soft buttons (named *a*,*b*,*c*,*x*,*y*,*z*) that the user's code can read.
  - All have *name*_**`button()`** functions which return 1 if the button is being pressed and 0 otherwise
  - All have *name*_**`button_clicked()`** functions which pause if the button is being pressed and then return 1 when it is released or return 0 otherwise
  - Soft buttons can have their display changed by using **`set_`***name*_**`button_text("`***display text* **`")`**
  - By default only a, b and c are displayed. The 3 extra buttons can be shown using:

```
extra_buttons_show()
extra_buttons_hide()
```

Demo

buttons and **if-else** statements (with **else if**)

```
int main(){
  set_a_button_text("1 Beep");
  set_b_button_text("2 Beeps");
  printf("press buttons for beeps, side button to stop\n");
  while (side_button() == 0){
    if (a_button() == 1){// can hold for continuous beeps
        printf("beep\n");
        beep();// beep flashes the screen
        msleep(500);
    }
    else if (b_button_clicked() == 1){// must release button
      // must press & release button before beeps happen
      printf("beep-beep\n");
      beep();
      msleep(300);
      beep();
    }
  }
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
```

What is a Function?

- Remember your math functions?
  - A typical function
    - Circumference of a circle is a function of the radius
      - The "circumference function" for a circle is the Greek circle constant π times twice the radius, or $C(r) = 2\pi r$
  - In general we use the notation f(x) to represent a function where f is the name of the function and x is its argument
    - Functions can have more than one argument, e.g., f(x,y)
  - Functions are "deterministic", meaning that if you supply values for the arguments, the function produces a unique result
    - $C(50) = 100\pi$ which is approximately 314.159

Functions in C

- A **C** program is comprised of 1 or more **C** functions, one and only one of which must be named **`main`**
- **C** functions follow the same rules as math functions, except a **C** function doesn't have to return a value and it doesn't have to have any arguments
- Since variables in **C** have differing types, you have to specify the data type for each of your function's arguments, and the type of data returned by the function (which can be **`void`** if nothing is being returned)

Function Prototype

- **C** expects you to specify a prototype for any functions before they are ever used
- A prototype for the circumference function would appear as:

function name *argument name*

```
double circumference(double rad);
```

data type returned *data type for the argument*

- In the KIPR Link help Manual the documentation for each KIPR Link library function gives the function's prototype

Using Circumference Function

```
// circumference function prototype
double circumference(double rad);

int main()
{
       /*This prints the circumference of a
       circle with a radius of 5.4 */
       printf("circumference is %g\n",circumference(5.4));
       // Note the function call is embedded in the printf
       return 0;
}

// circumference function definition
double circumference(double rad)
{
       return(2*3.1416*rad);
}
```

Demo

Previous Demo Converted to Use Functions

```
void beep_once();
void beep_twice();
int main() {
  set_a_button_text("1 Beep");
  set_b_button_text("2 Beeps");
  printf("press buttons for beeps, side button to stop\n");
  while (side_button() == 0) {
     if (a_button() == 1) { beep_once(); }
     else if (b_button_clicked() == 1) { beep_twice(); }
  }
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
void beep_once() {
  printf("beep\n");
  beep();
  msleep(500);
}
void beep_twice() {
  printf("beep-beep\n");
  beep();
  msleep(300);
  beep();
}
```

Previous Demo

```
int main(){
  set_a_button_text("1 Beep");
  set_b_button_text("2 Beeps");
  printf("press buttons for beeps, side button to stop\n");
  while (side_button() == 0){
    if (a_button() == 1){// can hold for continuous beeps
       printf("beep\n");
       beep();// beep flashes the screen
       msleep(500);
    }
    else if (b_button_clicked() == 1){// must release button
      // must press & release button before beeps happen
      printf("beep-beep\n");
      beep();
      msleep(300);
      beep();
    }
  }
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
```

Variables

- Just like arguments for functions, symbolic names can be used to retain data such as the current value of the distance traveled

  int distance; specifies a "variable" that can hold an integer

  – For the variable **distance** a program might use

  ```
  distance = get_create_distance();
  ```

  to store the value returned by **get_create_distance**
- Variable names in **C** are made up of contiguous letters (upper and lower case), the "_" character, and digits
  – Variable names cannot begin with a digit
  – Notice the practice of using "_" as a substitute for a space
- Variable types include **int, double,** and many others

Combining Time & Sensing

- If your robot uses msleep to drive for a specified time, it is literally "sleep moving" and will not be monitoring bumpers, buttons or other sensors
- The function seconds() returns a value of type double that represents the Link's internal clock.
- By getting the difference between the current value of seconds() and one stored from an earlier time, you can get the elapsed time and use that in your loop conditional.

Time & Sensing Example

- Here is a program that moves the Create for 5 seconds or until a bumper is pressed, whichever happens first

```
int main(){
  double start; \\will be used to hold the starting time
  create_connect();
  start=seconds;\\save the start time
  while((seconds()-start)<10.0 && //check the time & bumps
        get_create_lbump()==0 && get_create_rbump()==0){
    create_drive_direct(100,100);
  } \\exit loop when time is up or bumpers are bumped
  create_stop(); create_disconnect();
  return 0;
}
```

Activity 4 (Objectives)

Conditions and Functions

Write a program that behaves differently, dependent upon which software button is pressed. The code should be written using functions that you create for each of the different behaviors.

Activity 4 (Pseudocode)

Conditions and Functions

`main()`

1. Rename the A and C buttons to say Left and Right
2. Connect to the Create.
3. Loop while the side button is not pressed.
   - If the 'A' software button is pressed, then call the turn left function.
   - Otherwise, if the 'C' software button is pressed, then call the turn right function
4. Disconnect from the Create.

Activity 4 (Pseudocode)

Conditions and Functions

`turn_left(`**`double`** *seconds***`)`**

1. Move the Create left wheel backward and the right wheel forward.
2. Delay the program for an amount of time equal to the parameter value.
3. Stop the Create.

`turn_right(`**`double`** *seconds***`)`**

Pseudocode for this function is left as an exercise

Solution is on next two slides; try before you look!

Activity 4 (Solution)

Conditions and Functions

```
/******* If A turn left, if C turn right (mirror behavior) *******/
int turn_left(double seconds);  // prototype for turn_left
void turn_right(double seconds); // prototype for turn_right
int main() {
  // 1. Rename buttons
  set_a_button_text("Left"); set_c_button_text("Right");
  printf("Side button to stop\n"); // announce
  create_connect(); // 2. Connect to the Create
  // 3a. Loop until the side button is pressed.
  while (side_button() == 0) {
    // 3b. If the 'A' button is pressed, then turn left
    if (a_button() == 1) {
        printf("turned left %i degrees\n", turn_left(1.0)); }
    // 3c. Else if the 'C' button is pressed, then turn right
    else if (c_button() == 1) { turn_right(1.0); }
  }
  create_disconnect(); // 4. Disconnect from the Create
  printf("All Done\n"); // Tell user program has finished
  return 0;
}
```

(cont'd next slide)

Activity 4 (Solution, Cont'd)

Conditions and Functions

```
/*Function definitions go below*/
int turn_left(double seconds) {
  int initial_angle = get_create_total_angle(0);
  //  Move left wheel backward and right wheel forward
  create_drive_direct(-300, 300);
  //  Delay for time equal to the parameter value
  msleep(seconds*1000);
  //  Stop the Create
  create_stop();
  //  return the angle turned left
  return(get_create_total_angle(0) - initial_angle);
}
void turn_right(double seconds) {
  //  Move the Create left wheel forward
  //    and the right wheel backward
  create_drive_direct(300, -300);
  //  Delay for time equal to the parameter value
  msleep(seconds*1000);
  //  Stop the Create
  create_stop();
}
```

Activity 4 (Experiments)

Conditions and Functions

- Modify solution program so the number of degrees during a right turn is printed
- Download your code and run your program on the KIPR Link connected to the Create

Activity 4 (Reflections)

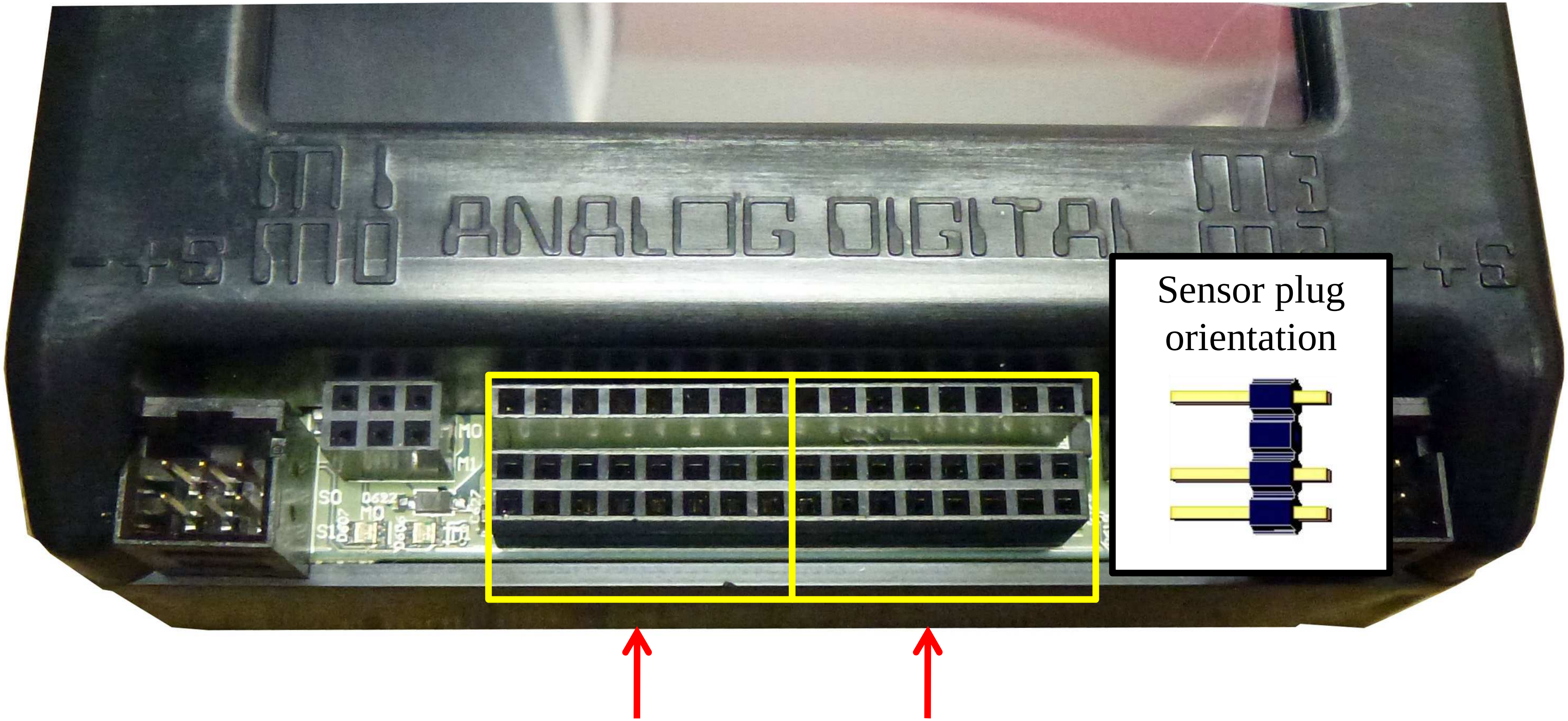
Conditions and Functions

- Does every **`if`** statement have to be succeeded by an **`else`** statement?
- Can you have **`if`** statements within other **`if`** statements?
- How many **`else if`** statements can follow an **`if`** statement?
- What is the difference between having an **`else`** and an **`else if`** statement at the end?
- Why is it useful to use functions?

Starting/Shutting Down the Robot Using Sensors

KIPR Link Sensor Ports

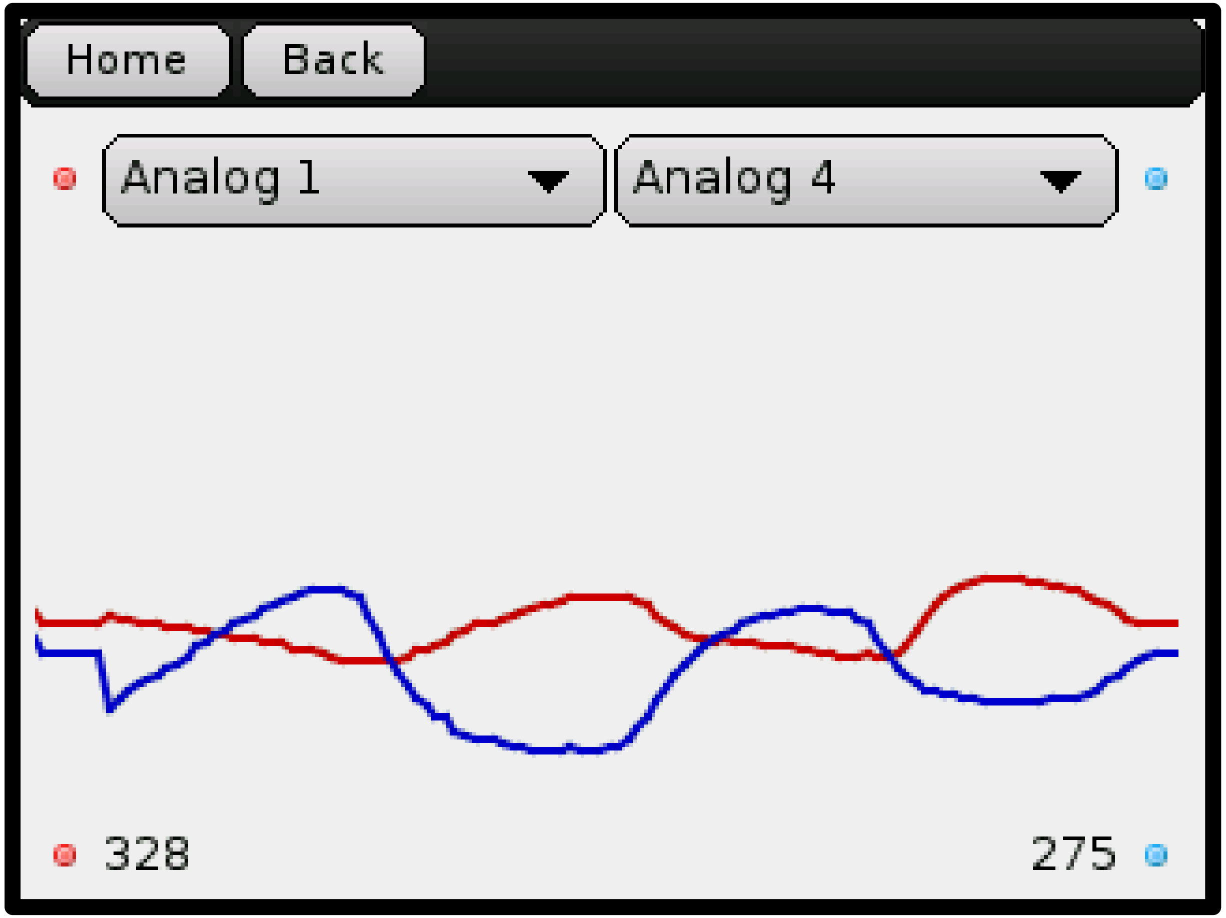
Analog and Digital



analog ports (0-7) and digital ports (8-15)

KIPR Link *Sensor Scope* Screen

- Go to the *Sensor Scope* screen
  - Under the *Motors and Sensors* tab on the opening screen, then under *Sensors*



KIPR Link *Sensor Scope* Screen

- Plug the two sensors to be used for this activity into analog ports on the KIPR Link
  - Plug the **light sensor** into any analog port (ports 0-7)

  - Plug the **reflectance sensor** into any other available analog port

- When you point the reflectance sensor towards the IR light sensor you should see a low value for its port
- If you aim the reflectance sensor at the table and move it across the table edge its value will change

Infrared Interlude

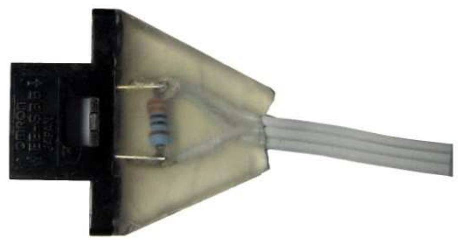
- Plug the USB camera into your KIPR Link and navigate to the Camera Page
- Point the camera at an infrared emitting sensor
- Most cameras are sensitive to infrared light
- You should see a lighted spot where the sensor's emitter is located
- Your light sensors can detect the emissions from the reflectance sensor emitter

Sensors for Activity

Using the KIPR Link and sensors

- IR light sensor
  - Analog sensor (pull up)
  - Plug into any port 0-7

- Reflectance sensors
  - Analog sensor (pull up)
  - Plug into any port 0-7
  - Has an IR emitter and an IR detector
  - Light source for this activity



analog10()

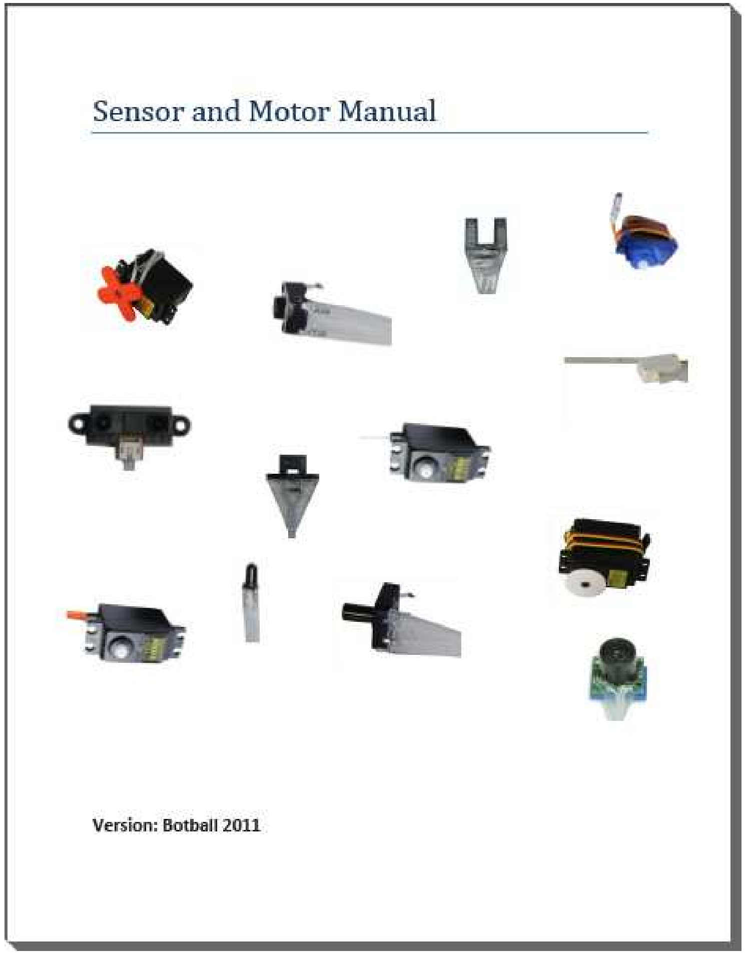
- For an analog sensor such as a light or reflectance sensor plugged into analog port 2, **`analog10(2)`** will provide the current value of the sensor
  - An analog sensor provides a range of values
  - The **`analog10`** function gives values from 0-1023

digital()

- For a digital sensor such as a button sensor plugged into digital port 8, **`digital(8);`** will provide the current value of the sensor
  - A digital sensor's value is 0 if off and ≠ 0 if on

Sensor and Motor Manual

- For further detail about sensors, consult the Sensor and Motor Manual on your workshop Team Home Base

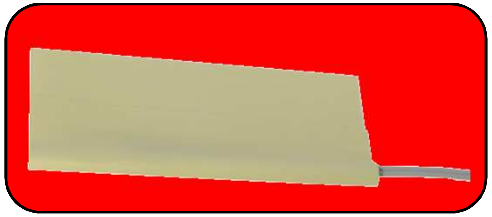


Shielding Light Sensors

A Botball Robot Should Have a Shielded Starting Light Sensor

- The table will be brightly lit
- Overhead lights from the game table will flood an unshielded starting light sensor rendering it incapable of discriminating ambient light from the starting light
  - Generally, to be effective IR light sensors should be shielded from all extraneous sources by a light tube
- Opaque objects stop IR light (e.g., foil, black electrical tape)
- Soda straws are not opaque; printer paper is not opaque; two layers of printer paper are not opaque; a straw wrapped in printer paper is not opaque

How to Shield a Light Sensor



No!!

Paper is <u>NOT</u> adequate shielding

1

2

3

Yes!

Slide straw over light sensor (leave a gap in the front) and tape in place covering the straw with electrical tape

`wait_for_light()`

- Botball tournament programs should start with the **`wait_for_light()`** library function
- The **`wait_for_light()`** function needs an argument (also called a parameter) which should be an analog port number
  - e.g., **`wait_for_light(3);`**
  - The KIPR Link has 8 analog ports (0-7)
- The **`wait_for_light`** function checks the value of the IR light sensor plugged into the port
- A low value indicates more IR (light on) is being detected, a high value less IR (light off)

Botball Robots Start …

- Botball robots have to start by themselves when the game table starting lights go on
- This requires determining the level of light at the table when lights are off and the level when the lights are on
- The **`wait_for_light`** function steps you through this calibration and then pauses until lights are on
- Crucial!!! you must use **`wait_for_light,`** or a version of it of your own creation, for Botball

Timing for Botball
`shut_down_in`

- When executed, the function

  `shut_down_in(`<*game_secs*>**`);`**

  starts a process that turns off all motors after *game_secs* has elapsed and keeps any new commands from being processed
- The **`shut_down_in`** function issues a **`create_stop`** command, but if your KIPR Link loses its serial connection to the Create (probably the result of a loose cable or an error in your program code), your Create won't receive the **`create_stop`** (and so won't stop in time, in which case you will lose the round!)

Activity 5 (Objectives)

Starting / Shutting Down the Robot Using Sensors

Write a program that monitors a light sensor and automatically moves the robot once light is detected.

- The robot should automatically turn off after a predetermined amount of time.

Run the program on the KIPR Link using the Create platform.

Demo

wait_for_light

```
/*********************************************************
*********  wait_for_light demo
*********************************************************/

int main()
{
  wait_for_light(0); // light sensor in analog port 0
  printf("I have seen the light!\nAll done\n");
  return 0;
}
```

Demo

shut_down_in

```
/*****************************************************
*********  shut_down_in demo
*****************************************************/
int main()
{
   printf("Program stops in 3 sec\n");
   shut_down_in(3.0);
   while (side_button()==0)
   {
       beep();
       msleep(200);
   }
   printf("All done\n");  // shuts down before this!
   return 0;
}
```

Activity 5 (Pseudocode)

Starting / Shutting Down the Robot Using Sensors

1. Monitor light sensor.
2. Move robot when light detected.
3. Have robot automatically shutdown after a certain amount of time.

Activity 5

Starting / Shutting Down the Robot Using Sensors

- Implement a program that follows the pseudocode
- The reflectance sensor contains an emitter and can be used as the light source for simulating the start light. Any time the start light should be on, shine the reflectance sensor at the light sensor (if there is a bright light available, the reflectance sensor is not needed)
- A solution is on the next slide if you need it

Activity 5 (Solution)

Starting / Shutting Down the Robot Using Sensors

```
/*****************************************************************
*********  Starting/shutting down the robot using the sensors
*****************************************************************/
int main() {
  printf("Activity 6\n");
  // 1. Connect to the Create
  while (create_connect());

  // 2. Wait for the light sensor on analog port 0 to turn on
  wait_for_light(0);

  // 3. Shut down in 5 seconds
  shut_down_in(5.0);

  // 4. Drive each of the Create motors at 100 mm/sec
  while (side_button()==0) {create_drive_direct(100, 100);}
  printf("All done\n");  // shut down before getting here!
  return 0;
}
```

Activity 5 (Experiments)

Starting / Shutting Down the Robot Using Sensors

- Increase the amount of time that the robot is active before it shuts down, and make the robot go straight and then turn during this time
- Add a reflectance sensor that points downward. Have the robot stop when this sensor detects a black line

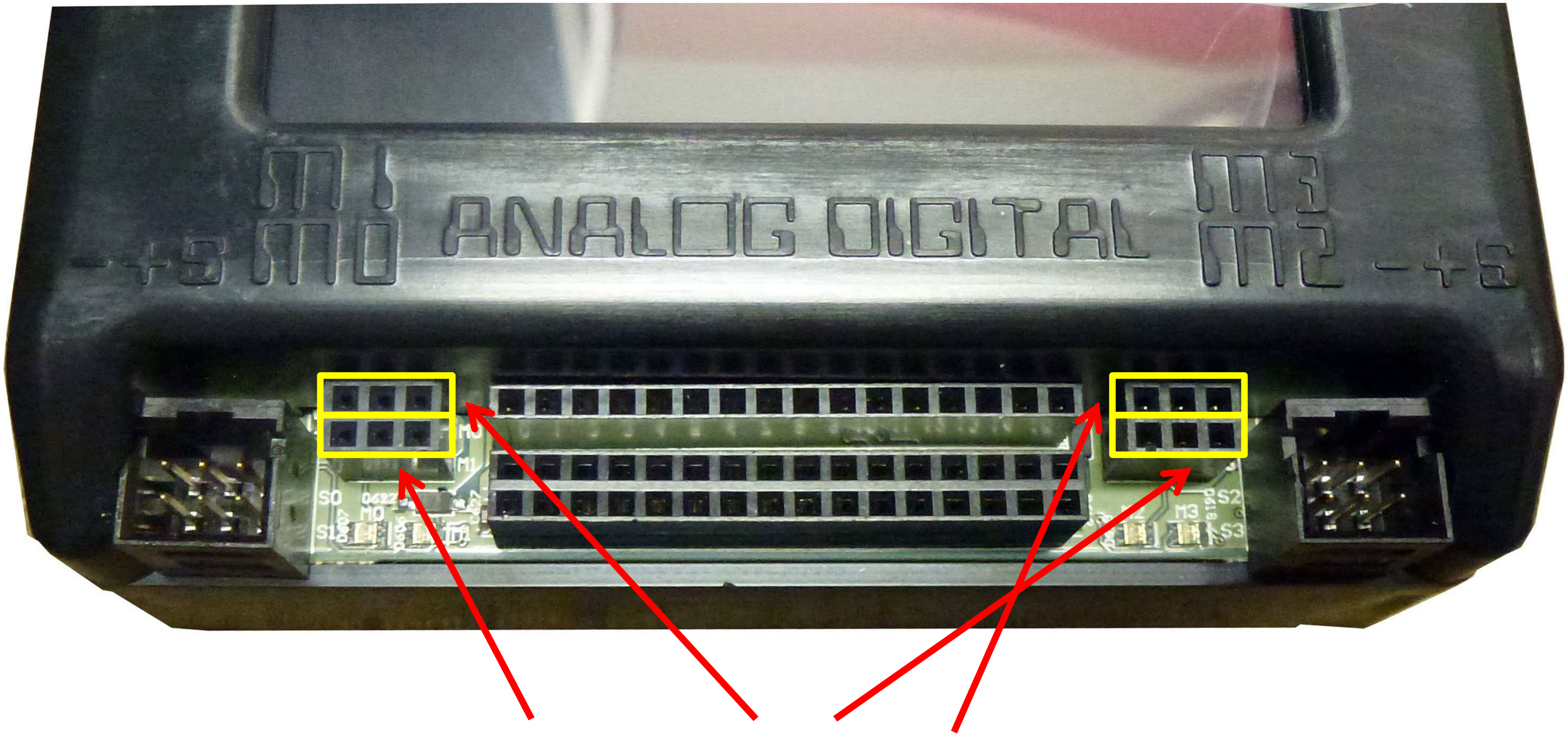
Activity 5 (Reflections)

Starting / Shutting Down the Robot Using Sensors

- In your own words, describe what the **`wait_for_light`** function does. How is this useful for the Botball competition?
- Describe what the **`shut_down_in`** function does? Why is it important that you use this function in your robot during the competition?
- What are the differences and similarities between analog and digital sensors?

Motors & Servos

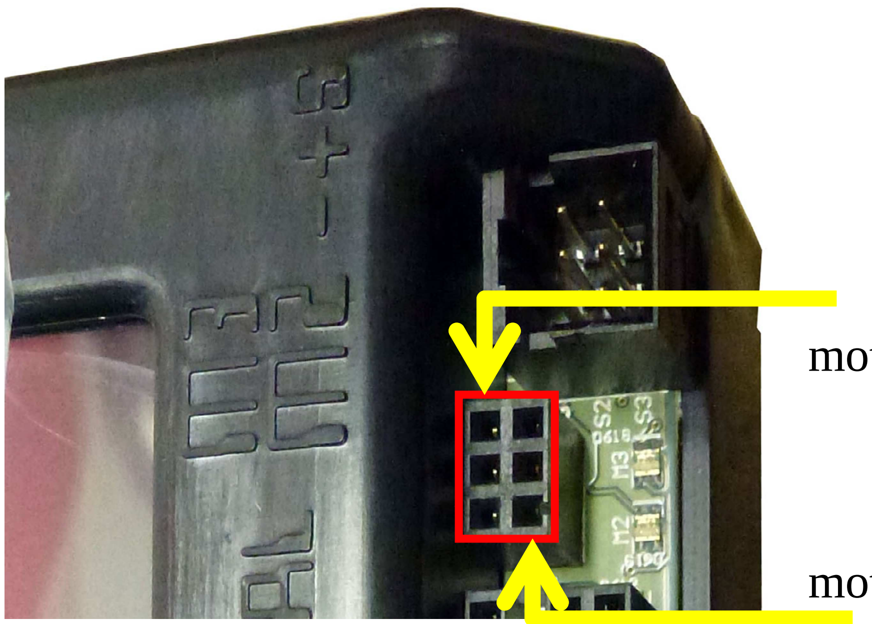
KIPR Link Motor Ports



Motor ports 0 (Demobot), 1, 2, and 3 (Demobot)

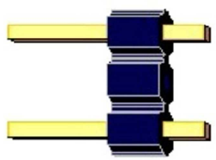
Plugging in DC Drive Motors

- DC drive motors are the ones with two-prong plugs and gray wires
- The KIPR Link has 4 drive motor ports numbered 0 & 1 on the left and 2 & 3 on the right
- When a port is powered it has a light that glows green for one direction and red for the other
- Plug orientation order determines motor direction, but by convention, green is forward and red reverse



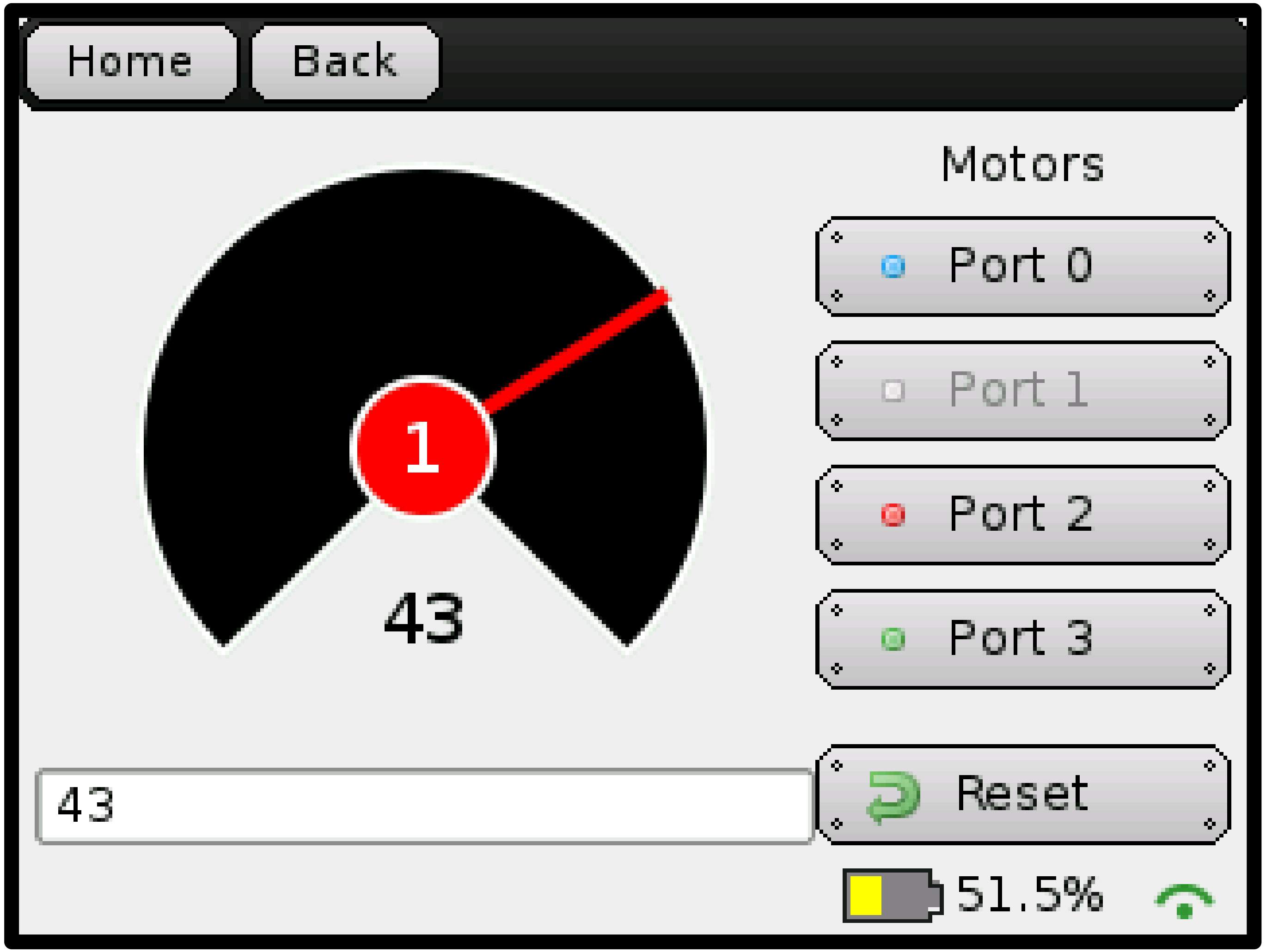
motor port 3

motor port 2



Drive motors have a 2 prong plug

KIPR Link *Motor PWMs* Screen



KIPR Link Motor Commands

`mav()`, `ao()`, `off()`

- **`mav(`***motor#*`,`*vel***`);`** [mav=move at velocity]
  - *motor#* is the motor port (0-3) being used
  - *vel* is the rotational speed of the motor measured in ticks per second (-1000 to 1000)
  - the amount of rotation per tick depends on the kind of motor
  - the motor runs at the set speed until turned off or commanded otherwise
- **`ao();`** turns off all motor ports
- **`off(`***motor#***`);`** turns off the specified motor port

Motor Position Counter

- As a DC motor runs, the KIPR Link keeps track of its current position in ticks
  - **`get_motor_position_counter(`***motor#***`);`**

    is a library function that returns this value for *motor#*
  - **`clear_motor_position_counter(`***motor#***`);`**

    is a library function that resets the *motor#* counter to 0
  - You can see the current value of the counter for a motor on the *motors..test* and *Sensor Ports* screens

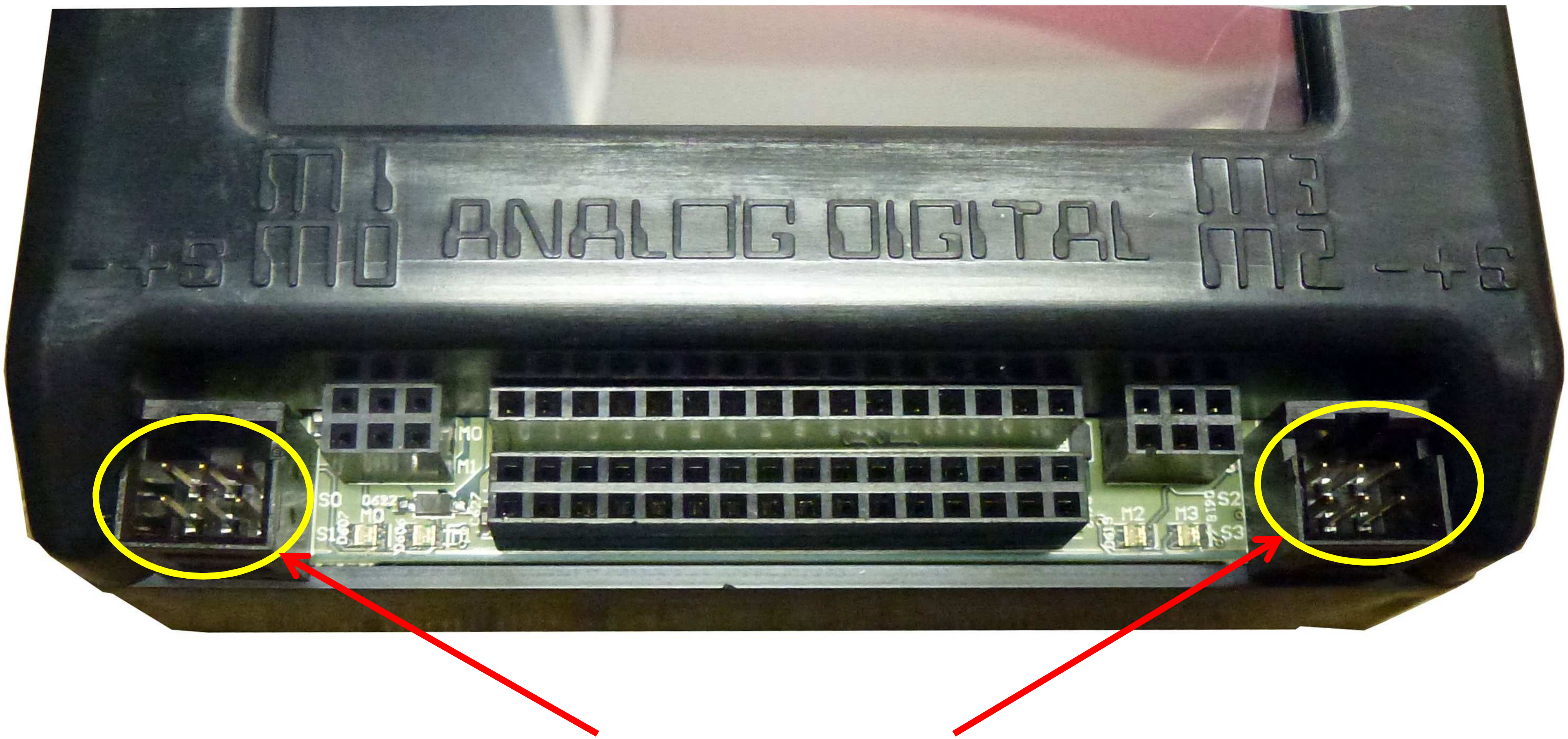
Motor Polarity

- Plug the drive motors into KIPR Link motor ports 0 and 3 (corresponding to simbot when running a program in the simulator)
  - Motor port numbers are labeled on the case below the screen
- Check motor polarity
  - Manually rotate each motor and observe its power light (it will glow red or green as you rotate the motor)
  - If a motor does not turn in the direction you want to correspond to forward (green), reverse its plug

Servo Motors

- A servo is a motor designed to rotate to a specified position and hold it
- To help save power, servo ports by default are not active until enabled
- A command is provided in the KIPR Link library for enabling (or disabling) all servo ports
  - **`enable_servos();`** activates all servo ports
  - **`disable_servos();`** de-activates all servo ports
- **`set_servo_position(2,925);`** rotates servo 2 to position 925
  - Position range is 0-2047
  - You can preset a servo's position before enabling servos so it will immediately move to the position you want when you enable servos
  - Default position when servos are first enabled is 1024
- **`get_servo_position(2);`** provides the current position of servo 2
  - Works only when servos are enabled

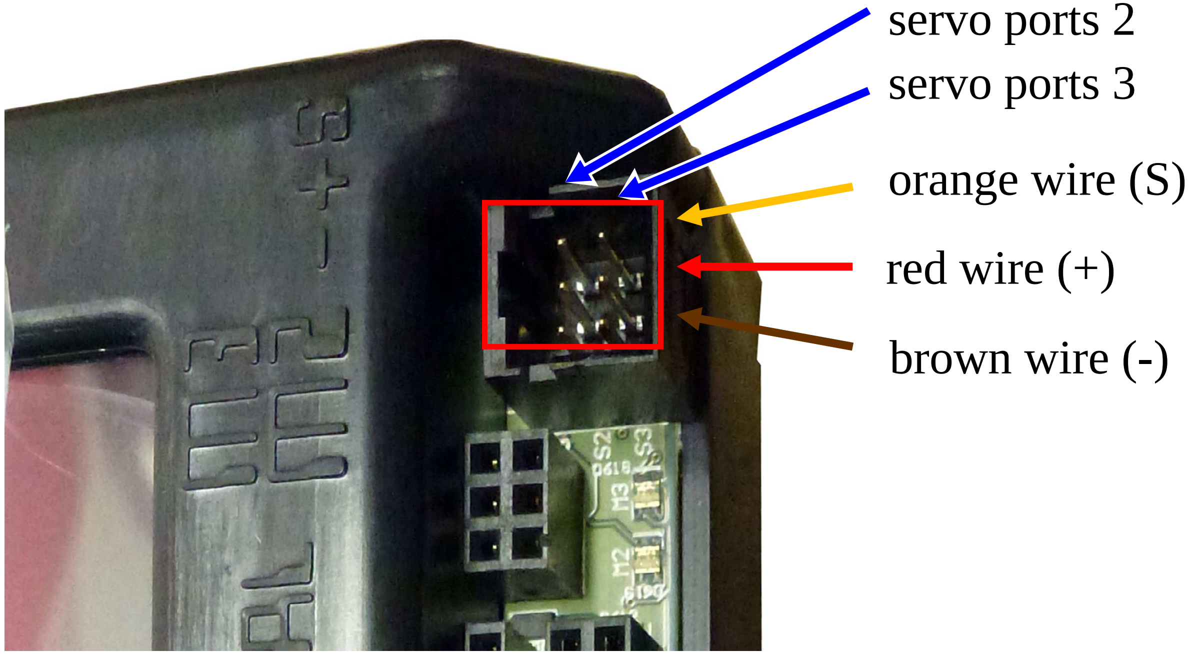
KIPR Link Servo Motor Ports

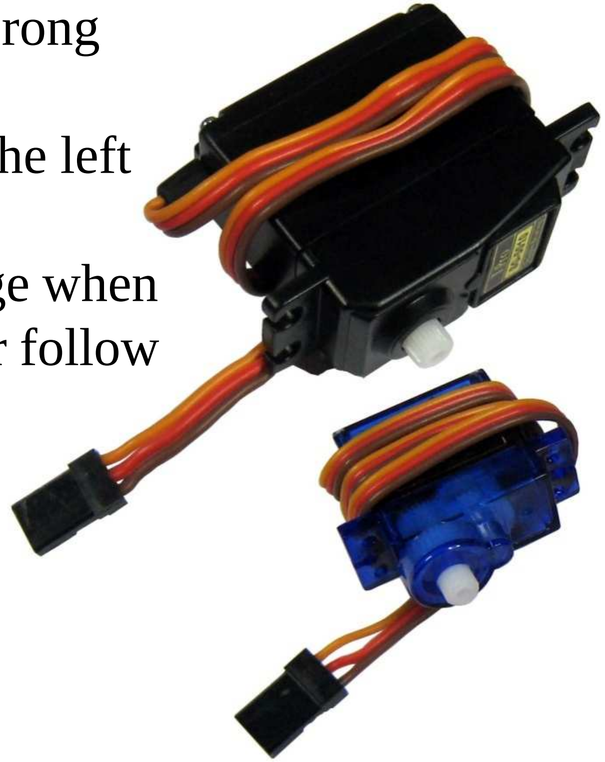


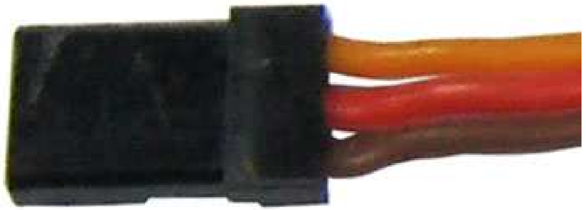
servo ports 0 and 1; servo ports 2 and 3

Plugging in Servos

- Servo motors (brown/black-red-yellow cables with 3 prong receptacle) plug into the KIPR Link servo ports
- The KIPR Link has 4 servo ports numbered 0 & 1 on the left and 2 & 3 on the right
- Plug orientation order is, left to right, brown-red-orange when the KIPR Link is oriented so the screen can be read (or follow the labeling: - + S; the orange signal wire goes in S)

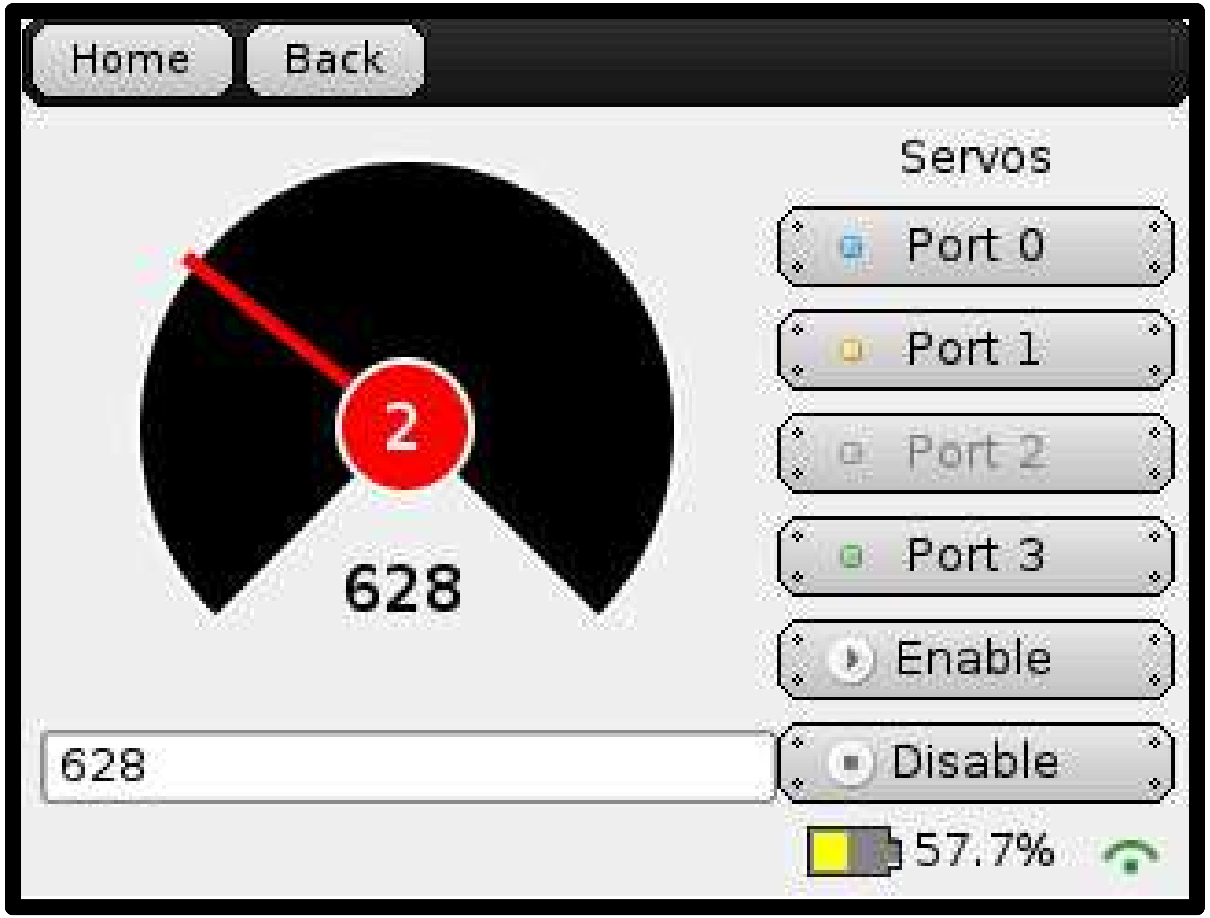






Botball®

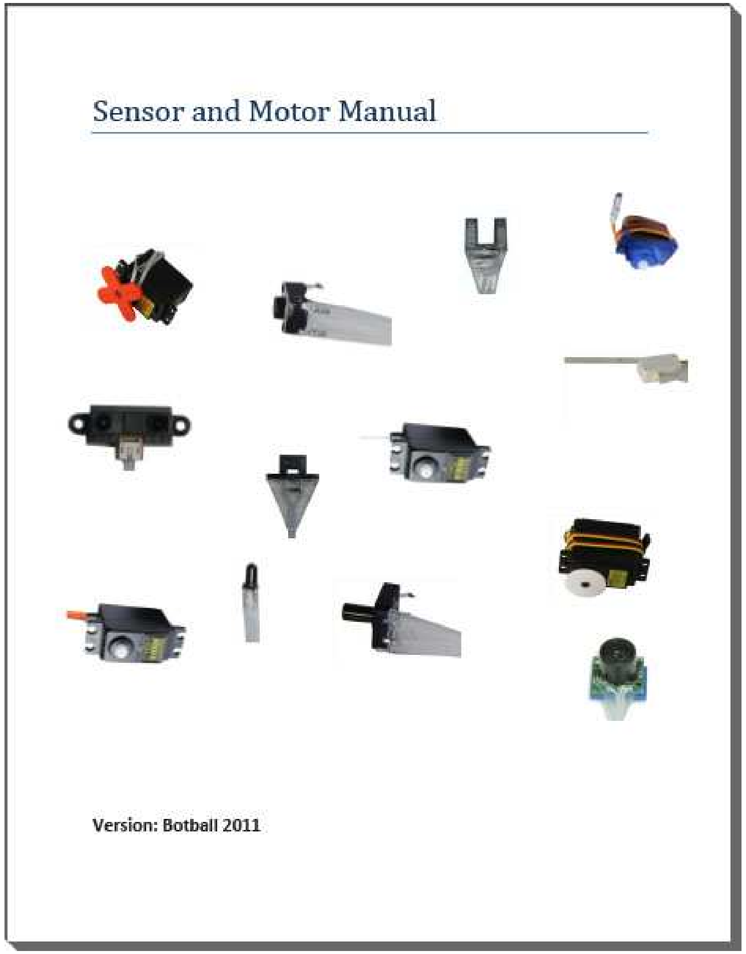
KIPR Link *Servo* Screen



The KIPR Link *Servo Test* screen can be used to center a servo or determine what position values to use once the servo is installed on a bot

Sensor and Motor Manual

- For further detail about motors, consult the Sensor and Motor Manual available via KISS IDE help



while Loop Operating a Servo

- A loop is a program construction used to repeat program steps until some condition is met
- Suppose we want to have a servo move from position 200 to position 1800 in steps of 100
  - we could do this by writing 16 separate **set_servo_position** commands after starting with **set_servo_position(1,200);**
  - with less effort, this can be done by using a **while** loop

```
set_servo_position(1,200); // move servo 1 to position 200
msleep(100); // give servo time to move
 while (get_servo_position(1) < 1800)
{ // move servo 1 in steps of 100
  set_servo_position(1,get_servo_position(1)+100);
  msleep(100); // give it time to move
}
```


Botball®

while Loop in a Program

Operate the Demobot arm using the buttons

```
int main() {
   int s1Pos=1024;
   set_a_button_text("Down");  set_b_button_text("Quit");
   set_c_button_text("Up");
   set_servo_position(1, s1Pos); // preset servo 1 position
   enable_servos(); // turn on servo
   printf("Move servo arm up and down with buttons\n");
   while(!b_button()){  // move servo 1 in steps of 100
     if(a_button()==1){set_servo_position(1, s1Pos-100);}
     if(c_button()==1){set_servo_position(1, s1Pos+100);}
     s1Pos = get_servo_position(1);
     if(s1Pos > 1950){set_servo_position(1,1950);}
     if(s1Pos < 150){set_servo_position(1,150);}
     s1Pos = get_servo_position(1);
     if(a_button() != c_button()){printf("servo at %i\n", s1Pos);}
     msleep(200); // pause before next move
   }
   disable_servos();  printf("done\n");
   return 0;
}
```

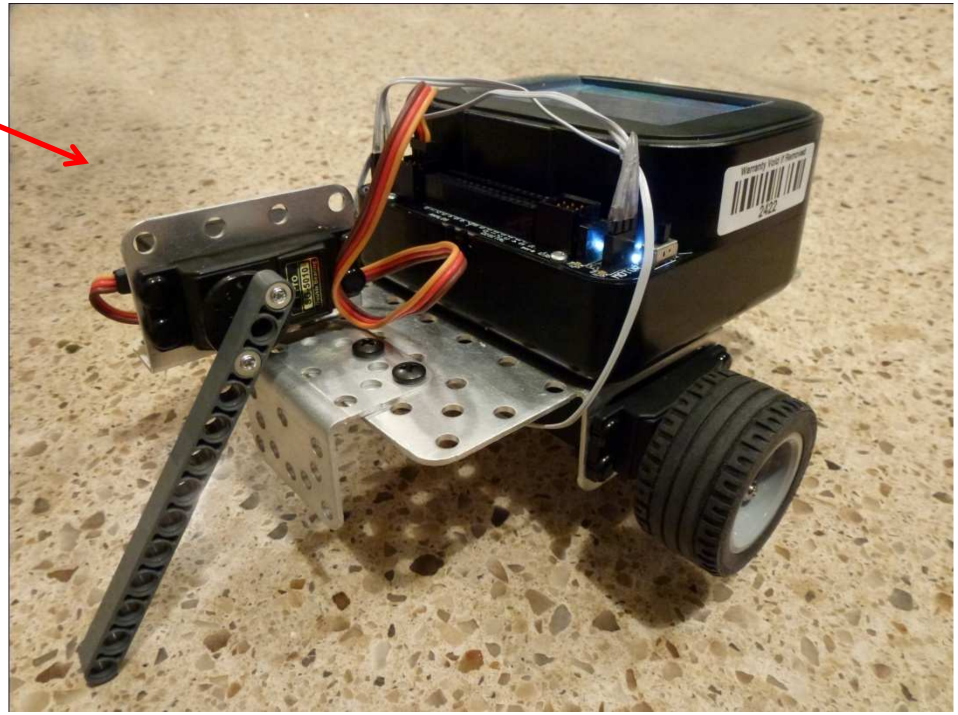
Activity 6 (Objectives)

Motors and Servos

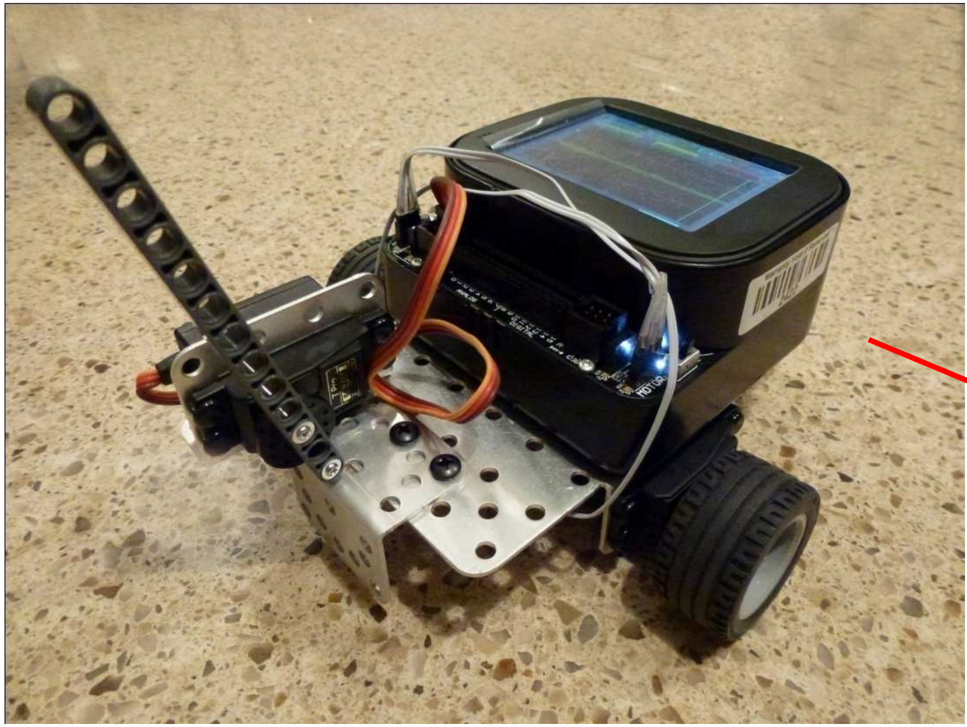
Lift the Demobot to a desired position using the servo and the accelerometer

Expected Behavior

After



Before



Botball®

Activity 6

Motors and Servos

- Have the robot detects when it is tilted, then stops the servo motion
- You should rely on the accelerometer values, not the servo position

Activity 6 (Solution)

Motors and Servos

```
/********  Stop when accelerometer shows robot has tilted
***************************************************************/
int main() {
   // preset servo 1 position
   printf("advance using A button\n\nB to quit\n");
   set_servo_position(1,200);
   enable_servos(); // turn on servos
   msleep(2000); // pause while it moves and user reads screen
   while((accel_y() > -150) && (b_button()==0))
   {  // move servo 1 in steps of 100
      set_servo_position(1,get_servo_position(1)+100);
      printf("servo at %d\n", get_servo_position(1));
      msleep(200); // pause before next move
      while((!a_button()) && (!b_button())) {}
   }
   disable_servos();
   printf("Tilt! Robot is done\n");
   return 0;
}
```

Activity 6 (Reflections)

Motors and Servos

- Why is the value of the B button being checked in each while statement?
- Why is the msleep statement before the second while?
- What does the robot do when disable_servos is executed?

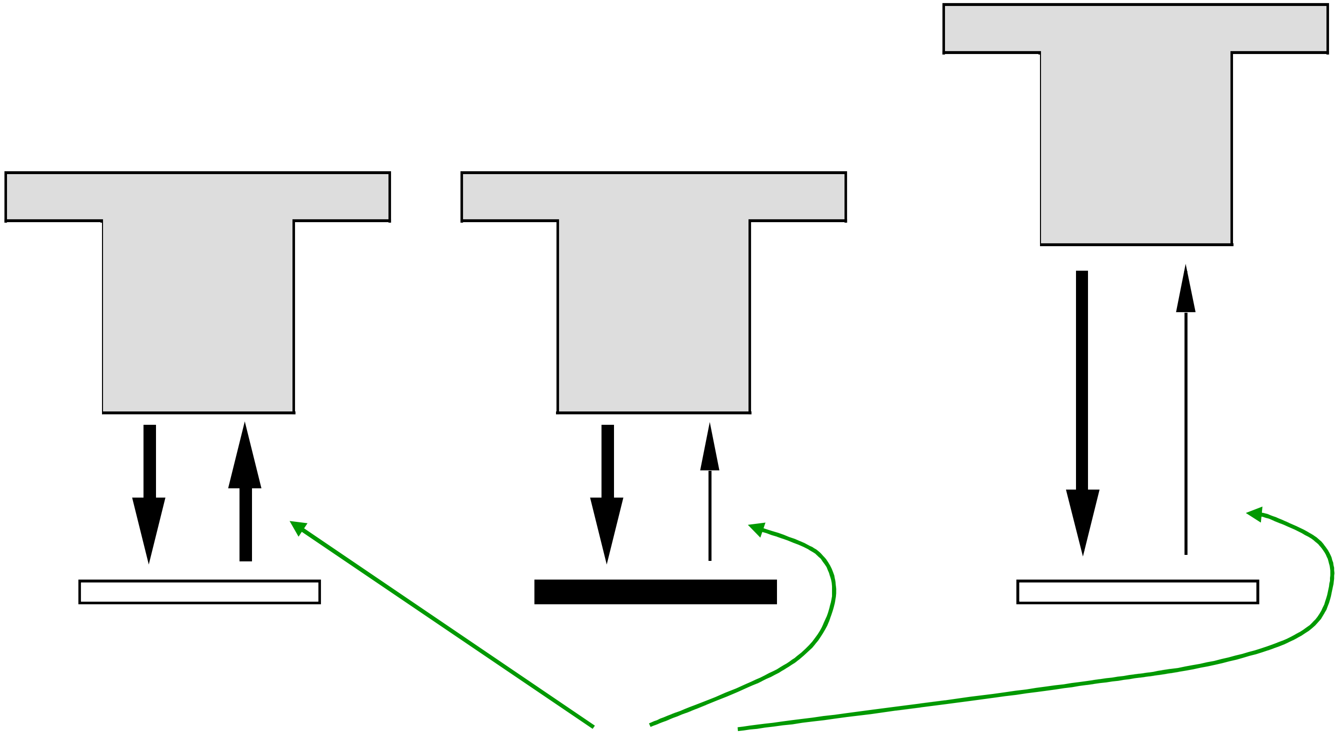
Analog and Floating Analog Sensors

IR Reflectance Sensors

- An IR reflectance sensor has an emitter producing an IR beam and an IR light sensor that measures the amount of IR reflected when the beam is directed at a surface
- There are two reflectance sensors in the Botball kit

- The KIPR Link library function **`analog10`** returns an amount that measures the amount of light reflected as a value between 0 and 1023
- A dark spot reflects less IR, producing a higher reading

IR Reflectance Sensor Behavior



Amount of reflected IR depends on surface texture, color, and distance to surface (higher values mean less IR indicating a dark surface or a drop off)

Demo

- Plug a reflectance sensor in port 0 and a light sensor in port 2

```
int main() {
  while(b_button()==0) {
    printf("reflectance: %i ", analog10(0));
    printf("light: %i\n", analog10(2));
    printf("B button exits\n");
    msleep(1000);
  }
  return 0;
}
```

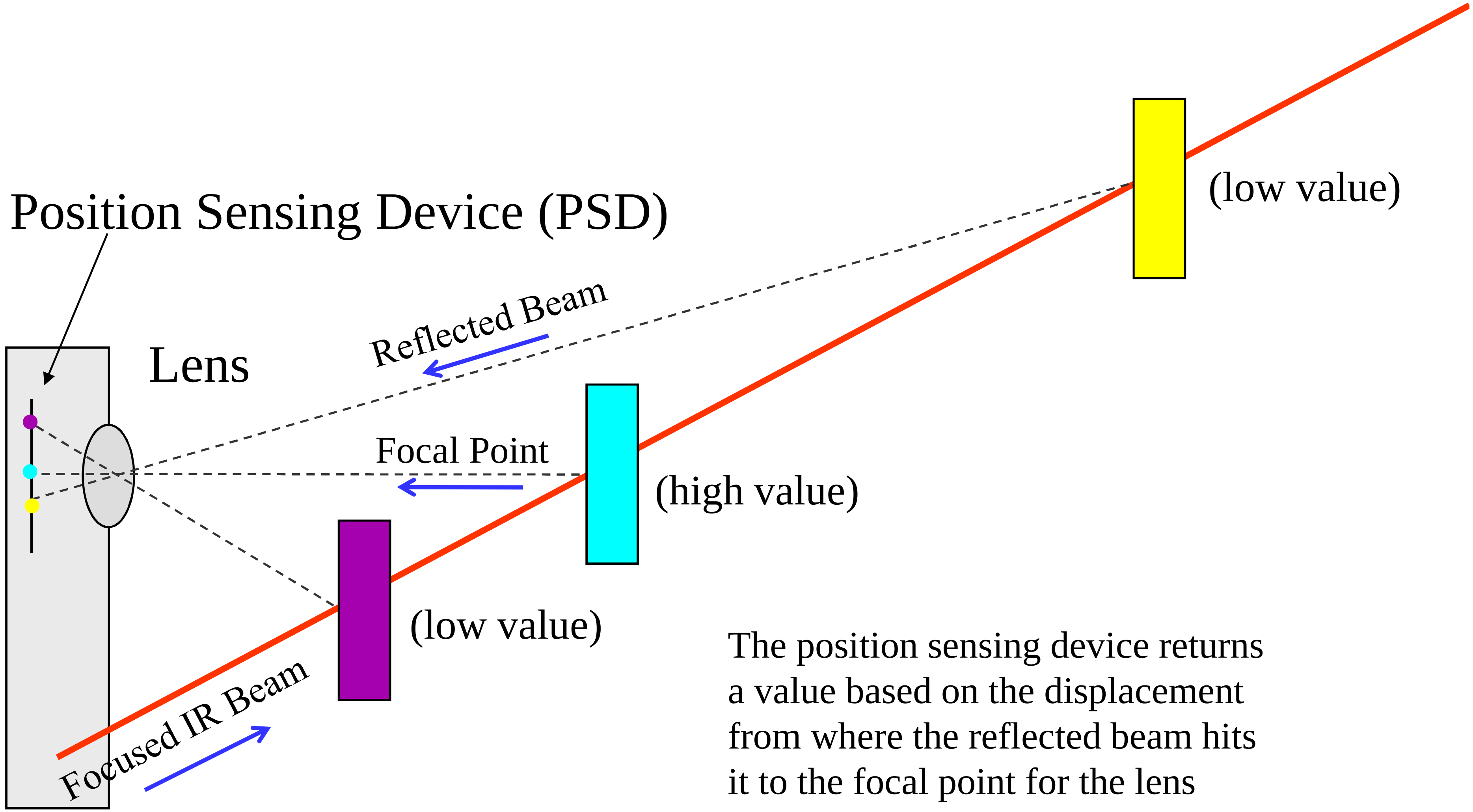
Optical Rangefinder "ET"

- Floating analog sensor
- Connect to ports 0-7 with pull-up disabled
- Access with library function **`analog10(`***port#***`)`**
  - You can also use **`analog(`***port#***`)`** for lower resolution
- Low values (0) indicate large distance
- High values indicate distance approaching ~4 inches
- Range is 4-30 inches. Result is approximately $1/d^2$. Objects closer than 4 inches will produce values indistinguishable from objects farther away

Pull-Up Resistors

- Most sensors need a pull-up resistor to register accurate values
  - Pull-up resistors are engaged by default
- Some sensors, e.g., the ET range sensor requires the port to be **floating**, i.e., have no pull-up resistor
- The KIPR Link can change each analog port to be either analog (pull-up resistor) or floating analog (no pull-up)
  - **`set_analog_pullup(3,0);`** sets port 3 to floating and leaves the other analog ports as they were

Optical Rangefinder



The position sensing device returns a value based on the displacement from where the reflected beam hits it to the focal point for the lens

Demo

- Plug an ET sensor in port 1

```
int main() {
  set_analog_pullup(1,0);
  while(a_button()==0) {
    printf("ET: %i (A Button Exits)\n",
            analog10(1));
    msleep(1000);
  }
  printf("All done\n");
  return 0;
}
```

Line Following

Objectives

Line following

- Have a robot follow a line it can detect using a reflectance sensor (attach with UGlu)

Prep

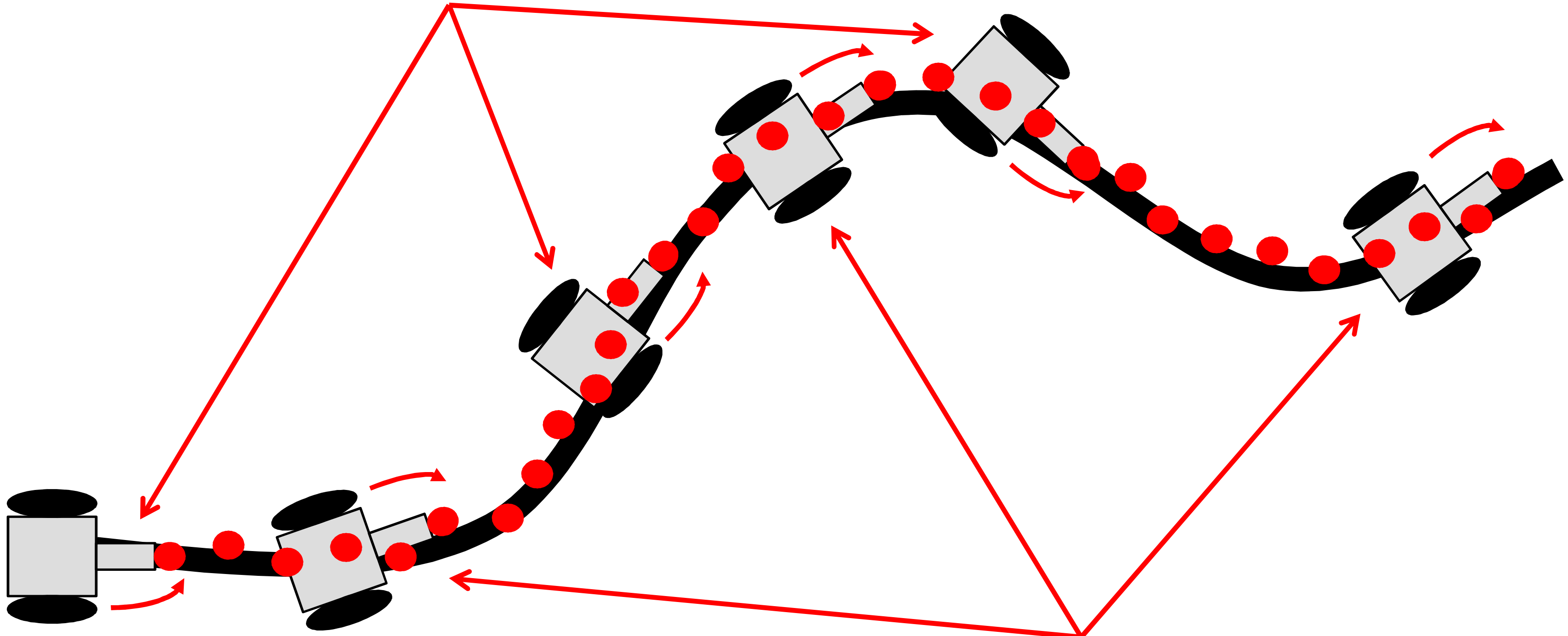
Line following

- Prep
  - Reflectance sensors
  - Line following strategies
  - Turning through an arc
  - Robot setup
  - Program steps for activity

Line Following Strategy

Follow the line's left edge by alternating the following 2 actions:

1. If detecting dark, arc left



2. If detecting light, arc right

Robot Setup

Line following

– Position your robot so the sensor is over the line and observe values on the *Sensor Screen* as you move the sensor left or right over the line

Program Steps

Line following

- Activity (DemoBot)
  - Starting with the sensor over the line have your program repeat the following two steps
    1. Until the sensor detects dark, turn in an arc left
    2. Until the sensor doesn't detect dark, turn in an arc right
        - the robot follows the left edge of the line
        - Each step requires a loop (indicated by the word until)

Activity 7

Line following

- Write a program to follow black tape line on a surface enough lighter than the tape that a reflectance sensor can tell whether or not it is over the tape
  - Have your program wait for the side button to be pressed to start following the line
- If you can't figure it out, there is a solution in a couple of slides

Reflections

Line following

- What happens if a turn in the line is too tight?
- What happens if there is a gap in the line?
- What happens when the robot reaches the end of the line?
- Only 1 reflectance sensor is being used for line following in this activity. What strategy using additional reflectance sensors might improve accuracy and/or allow you to go faster?

Activity 7 (Solution)

Line following

```
/* Line following with a single sensor: arc left when the
    reflectance sensor detects dark and otherwise arc right
 Use the Sensor Ports screen to find the high & low
    reflectance sensor values for the robot on the surface
 Set the threshold halfway between                        */
int main()
{
  int rport=7, leftmtr=0, rghtmtr=3;  // identify port and motors
  int threshold=512;          // set threshold for light conditions
  int high=100,low=-10;       // set wheel powers for arc radius
  printf("Line following: position robot on tape\n");
  printf("Press B button when ready\n\nPress side button to stop\n");
  while(b_button()==0) {}     // wait for button press
  while(side_button()==0){             // stop if button is pressed
     while (analog10(rport) > threshold) { // continue until not dark
       motor(leftmtr,low); motor(rghtmtr,high);   // arc left
       if (side_button()!=0) break; }     // or button pressed
     while (analog10(rport) <= threshold){     // continue until dark
        motor(leftmtr,high); motor(rghtmtr,low);  // arc right
       if (side_button()!=0) break; }     // or button pressed
  }
  ao();  // stop because button pressed
  printf("done\n");
  return 0;
}
```

Day 1 Homework

- Reuse Teams: Bring LEGO bricks and Create!
- Work on activities you didn't get to or come in early to work on them
  - Challenge: implement line following with the Create (the cliff sensors are reflectance sensors – see the KISS IDE help file for the function syntax)
- Thoroughly review the game slides on your Team Home Base
  - There will be no game review tomorrow, only a 30 minute Q&A
- Test sensors, motors, and KIPR Link ports
- Review the KISS IDE *Help*
- Review the manuals on your Team Home Base
  - KIPR Link Manual
  - Sensors and Motors Manual
- Review the BOPD Manual (Botball Online Project Documentation)
- Review "New items for 2013" on your Team Home Base
- Read the "Hints for New Teams Manual" on your Team Home Base
- Send your instructor any questions for the Day 2 recap (email or paper)

to Day 1

Botball 2013 Educators' Workshop Day 2

Botball 2013
Educator's Workshop
Day 2

1. Sign in
2. Robot controllers back on charge
3. Review Game rules from Team Home Base
4. Early arrivers verify DemoBot is ready and finish up from yesterday

Day 1 Slides

House Keeping

Day 2

- Recap: Introductions
- Daily schedule
- Take the survey at

  https://www.surveymonkey.com/s/HPBMJ6Z

Workshop Schedule

- Day 1:
  - Overview of Botball
    - Botball season, related events
    - Game preview/video
    - Resources & teams
  - Topics and Activities
    - Activity 0: The KISS IDE
    - Activity 1: Programming basics
    - Activity 2: Driving Straight
    - Activity 3: Build DemoBot
  - Lunch
    - Activity 4: Conditions and functions
    - Activity 5: Starting / shutting down the robot using sensors
    - Activity 6: Motors and servos
    - Activity 7: Line following
  - Homework
- Day 2:
  - New Team Suggestions
  - 30 minute game Q&A
  - BOPD
  - Continue with activities
  - Topics and Activities
    - Activity 8: Vision
  - Lunch
  - Selected activities
    - Activity 9: Point servo at colored object
    - Activity 10: Bang-Bang control
    - Activity 11: Proportional control
    - Activity 12: Approach specific QR code
    - Activity 13: Bang-Bang DemoBot arm
    - Activity 14: Proportional DemoBot arm
    - Activity 15: Accelerometer for bump detect
    - Activity 16: Music on the Create
    - Activity 17: Reduce heading errors

2013 Botball National Sponsors

NASA

ONR
Office of Naval Research

NORTHROP GRUMMAN
Foundation

iRobot®

igus®

3DS SolidWorks

COMMON SENSE RC

2013 Regional Botball Sponsors

ANALOG DEVICES

ARVEST

astellas
Leading Light for Life

BAE SYSTEMS

BancFirst
Loyal
To Oklahoma & You.

DoD Ordnance Technology Consortium
INDUSTRY ACADEMIA GOVERNMENT
NWEC
DOTC
PRAESTANTIA PER CONSORTIO

EQUAL FOOTING FOUNDATION

Oklahoma Aeronautics Commission

OU College of Engineering

FIRST LEGO LEAGUE | BOTBALL | VEX ROBOTICS
FIRST ROBOTICS | UNDERWATER ROV | MICRO ROBOTICS
ROC
ROBOTICS ORGANIZING COMMITTEE
HAWAI'I

HITACHI
Inspire the Next

QUALCOMM

SDSA
San Diego Science Alliance

PROJECTION
PRESENTATION TECHNOLOGY

Rockwell Collins

Rambus

TFCU
Return to the Home Page

OKLAHOMA STATE DEPARTMENT OF EDUCATION

KIRKPATRICK FOUNDATION

Botball

Regional Workshop & Tournament Hosts

UNF UNIVERSITY of NORTH FLORIDA

TU WIEN TECHNISCHE UNIVERSITÄT WIEN Vienna University of Technology

SOUTHERN ILLINOIS UNIVERSITY EDWARDSVILLE

APL STEM JOHNS HOPKINS UNIVERSITY Applied Physics Laboratory

USC University of Southern California

NASA

S·T·E·M Outreach
A MITRE Corporation Initiative

NM STATE UNIVERSITY

GROSSMONT COLLEGE

UMASS LOWELL

WORCESTER STATE UNIVERSITY

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Georgia Institute of Technology

LOYOLA UNIVERSITY CHICAGO
AD · MAJOREM · DEI · GLORIAM 1870
Preparing people to lead extraordinary lives

HAWAI'I
HAWAIʻI CONVENTION CENTER
WHERE BUSINESS AND ALOHA MEET™

MESA Mathematics Engineering Science Achievement

THE UNIVERSITY OF ARIZONA
Arizona's First University.

MOREHOUSE COLLEGE
PROJECT IDENTITY
A Title III Program

UNIVERSITY OF MARYLAND

Botball®

GCER 2013

The 2013 Global Conference on Educational Robotics will be held at the Embassy Suites in Norman, Oklahoma from **July 6-10, 2012** with preconference classes on July 5th

Global Conference on Educational Robotics
http://www.kipr.org/gcer

Conference events will be held onsite in the conference facilities. We have a discounted block of rooms at Embassy Suites and strongly suggest you stay onsite.
http://www.kipr.org/gcer/housing

We have secured special rates, which include breakfast and a daily manager's receptions for adults

Botball®

Global Conference on Educational Robotics

ALL TEAMS ARE INVITED!

When

– July 6th – July 10th
– Pre-conference activities and workshops July 5th

Who

– Middle school and high school students, educators, robotics enthusiasts, and professionals from around the world

Activities

– Meet and network with students from around the country and world
– Talks by internationally recognized robotics experts
– Teacher, student, and peer reviewed track sessions
– International Botball Tournament
– KIPR Open Tournament (Botball for grown-up kids!)
– Autonomous Robotics Showcase

New Teams

These are helpful hints and suggestions but by no means the only way to implement and manage your program

Right After the Workshop!

1. **<u>Recruit team members.</u>**
   If you haven't already recruited team members you can use the game video from the workshop to show to interested students.

2. **<u>Hit the ground running.</u>**
   - Do not wait to get started.
   - You only have a limited build time before the tournament and time is of the essence.
   - The workshop will still be fresh in your mind if you start now.
   - Plan on meeting sometime during the **<u>first week</u>** after the workshop**.**

Right After the Workshop!

Students will not inherently know how to mange their time, lets face it, it is hard for many adults!

3. **Plan out the season**

- Mark a calendar or make a Gannt chart with important dates:
  - 1st submission documentation due
  - 2nd submission documentation due
  - 3rd submission documentation due
  - Tournament date
- Set dates and schedules for team meetings
- Plan on meeting a **minimum** of 4 hours per week. (Botball teams average 8 hrs/week nationwide)
- Team meetings can be held with the first order of business being going over the calendar and any upcoming due dates

Right After the Workshop!

3. **<u>Plan out the season (continued)</u>**

- A **large** calendar or project plan displayed where everyone can see it is a good way to go.
- You can draw one on your whiteboard (If the janitor doesn't erase it) or put it on butcher or poster paper.
- The local lumber supply store (Lowes or Home Depot) will carry 4' X 8' sheets of melamine backed 1/8" Masonite, that is relatively inexpensive (~$12). You can write on it just like a whiteboard, using a permanent marker for the grid and whiteboard (erasable) markers for everything else. It can easily be cut into smaller sizes and mounted on the wall.

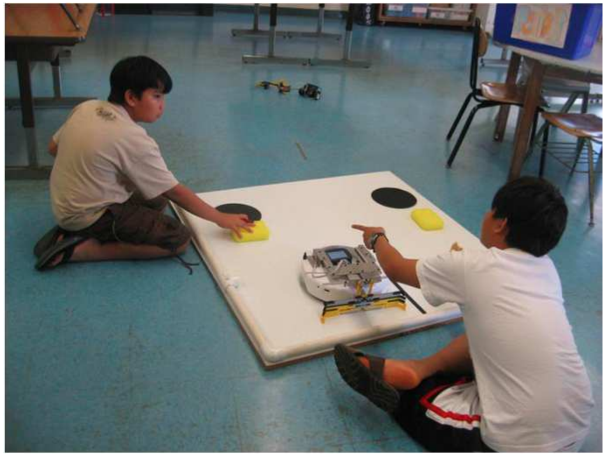
Right After the Workshop!

4. **Build the game board**
   - The material list and construction instructions for the KIPR tournament setup are on the **team home base**.
   - This can be a great parent, mentor and student activity.
   - The cost is ~ $100, but you can reuse the expensive FRP for next year's game board.
   - The board is designed so that you can take it down and put it back up in your classroom.
   - Many teams have a classroom or another room in the school where they can leave it set up. Your school may or may not have another room you can use, but it doesn't hurt to ask.

Right After the Workshop!

5. <u>If you can't build the full game board</u>

- You can build ½ of the board.
- You could tape the outline of the board onto a floor if you have the right type of flooring.
- You might be able to talk with another team who does have a board that they would let your team use on a practice day. If you are in this position and don't know whom to contact, call us and we can make introductions and see if something can be set up in your area.



Right After the Workshop!

6. **Kit Organization**

- Organized parts can lead to faster & easier construction and redesign of robots.
- Tupperware® containers, tackle boxes, anything that keeps the parts organized.
- This makes it easier to lock or move the components when you have another class or are not working on the robots, including transporting everything to the tournament.
- If a part breaks, it is easier to find a replacement.
- This is a good job for team members and will help them learn what is in the kit by sorting and counting.

Right After the Workshop!

6. <u>Kit Organization (continued)</u>

- Tupperware® containers or cardboard boxes are great for holding the robots in progress.
- Allows for easier transport to the tournament.
- You can keep the robots from distracting other classes.
- You can keep the robots safe.
- REMEMBER- There are no requirements to use all of the parts included in the kit.

Right After the Workshop!

6. Understand the Game

- This is what you should go over with your students on the first meeting after the workshop.
- Go over the game by using the game table you have built or by drawing the game field on the board or by projecting the game field (on the team home base) onto a screen.
- The goal is to have students identify game pieces and areas on the board where points are scored.
- The game board has markings to help team's robots navigate or locate their position.
- If it is on the board there is a reason for it. For example: A black line leading from the starting box to the scoring area could be used by a line following program.

Right After the Workshop!

6. Understand the game (continued)

- KIPR always includes multiple tasks that score points.
- **There is always a relatively easy way to score points. DO NOT overlook this**.
- Many teams are very successful because they get the simple, "easy points" consistently every time their robot(s) run.
- Focus on one behavior/task at a time
- Start with easy tasks and work your way to the more difficult ones
- **Remember, scoring a few points consistently and having success is better than being unsuccessful and scoring no points.**

Don’t forget about the resources!

The Link, sensor and motor manuals contain a lot of useful information.

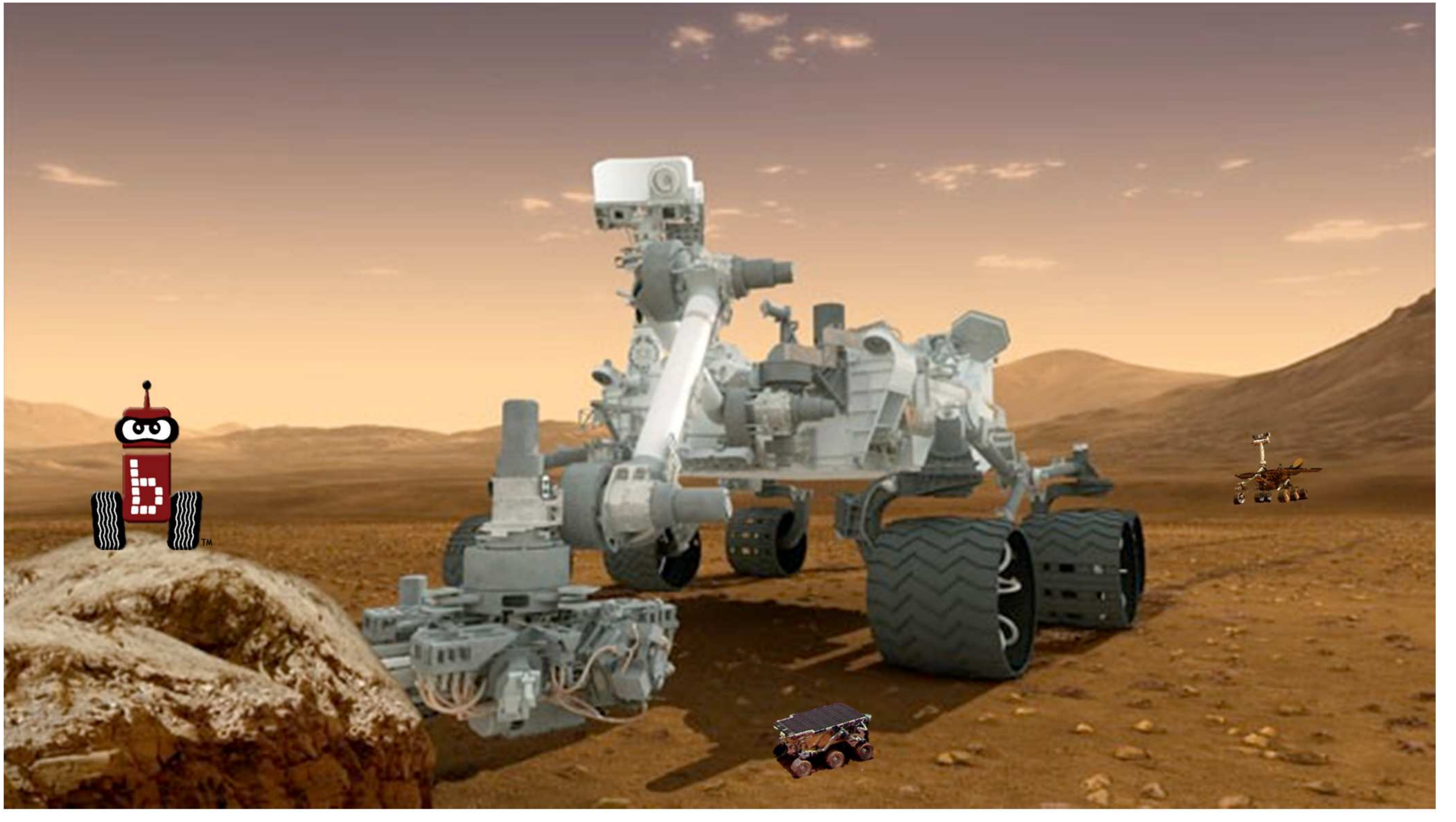
- They are electronic, but some teachers choose to print them out and put them in 3 ring binders for easy access by the students.
- This is also true of the game rules & specifications and the documentation requirements.
- This can be a great student activity and it gives you an easy answer when students ask a question pertaining to those topics; “Did you look in the binder?”
- Use the construction hints (pictures of previous robots and robot subsystems) on the team home base.

Ideas on Construction

It is important to note that our competition tables are built to specifications with allowable variance.

- **DO NOT** engineer robots that are so precise a 1/4” difference in a measurement means they are not successful. For example: the specified height of the elevated platform is15”, but at the tournament the platform could actually measure 15 1/8”. If your arm is set for exactly 15” it will not work.
- Review construction documents (like the ones on the Home Base) to get building ideas
- Search the internet for other robots and structures to get building ideas
- Test structure robustness before the tournament

You are not alone!



Communication with KIPR

- **Newsletters-**These have important information-check your spam filter.
- **Emails-** These include important information so read them and forward them to your team-check your spam filter.
- **Team Home Base-** Check this often, especially the FAQ for rule questions and technical solutions.
- You can call KIPR staff/**technical support** during office hours 8:30-5:00 pm CT 405-579-4609.
- You can also **email** **support@kipr.org** any time.
- Use the **community site** http://community.botball.org/
- Programming tutorial http://nasarobotproject.wordpress.com

A word about Documentation

- What?
  - Botball Online Project Documentation
- When?
  - 3 periods during design and build portion
  - 1 onsite presentation (8 minute) at regional tournament
- Why?
  - To reinforce the engineering process
  - **POINTS EARNED IN DOCUMENTATION FACTOR INTO OVERALL TOURNAMENT SCORES!**

See BOPD Handbook on the Team Home Base for more information (rubrics and exemplars)

When you come to the tournament.

- Adults are NOT allowed in the pits (you can help them get settled in and then you must leave)
- Bring ALL of your equipment. Especially your charging cables, extra LEGO etc.
- Plan on staying for the awards (There are a lot of Judge’s Choice Awards)

About the starting lights….

- The competition game board will have two moveable lights on each side.

About the starting lights….

- The competition game board will have two moveable lights on each side.
- All robots must use a light sensor and be programed to **wait for the light** and then start autonomously.
- Robots must shut down automatically at the end of the match.
- If students do not understand and accomplish this, the robots will never start or they will not shut down and they will be disqualified.
- After calibration, do **<u>not</u>** move the robot or light sensor
- MAKE SURE your students understand how to shield and mount their light sensors, adjust the starting lights and calibrate them prior to the tournament.

Management Ideas

Recruit some help.

- If possible, recruit another teacher or parent to help out.
- Parents do not have to be engineers or programmers to help.
- Someone to help organize, bring snacks, sit in the classroom, oversee students and keep them on task can be a big help.

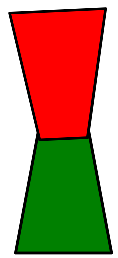
Divide and conquer.

- You have two robots to design, build and program divide the team between the two robots.
- Don't forget about the documentation.
- Divide robots performance tasks into subtasks and use them as assignments.
- Facilitate- Keep them in check on goals, time lines and expectations. Team meetings are great for this.

Management Ideas

Herding Cats

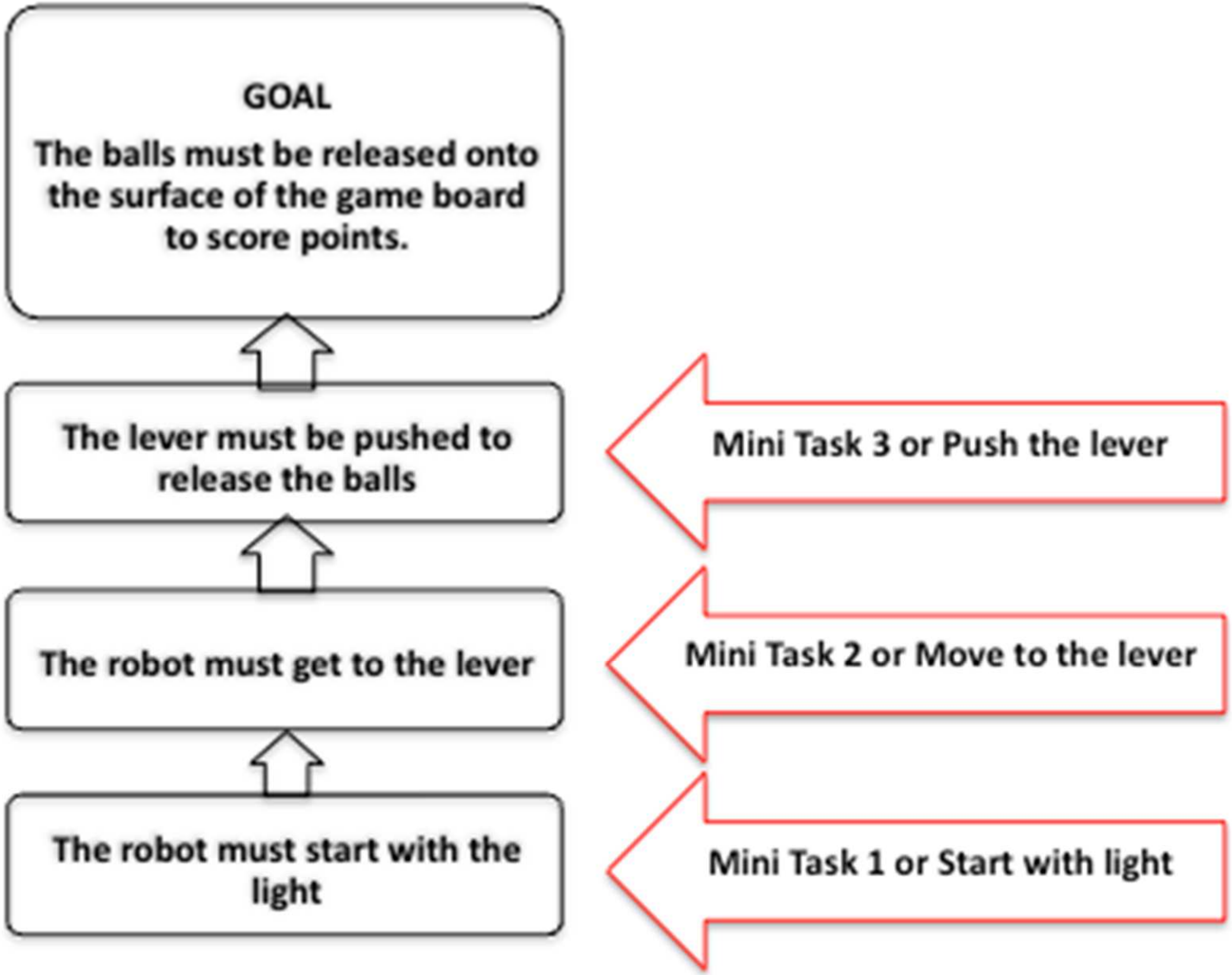
- Many students will need a lot of help and practice working independently (Not running to you with every question and problem)
- Set a policy/procedure of – "before coming to me did you …" (check the resources, ask your team mates, look online, etc)?
- Use the green cup/red cup.
  - Tape a green and red cup together (base to base) for each group.
  - The group leaves the green cup UP if they have no questions or issues.
  - They turn the red cup UP if they have gone through the policy/procedure and still need help.



How do I teach this?

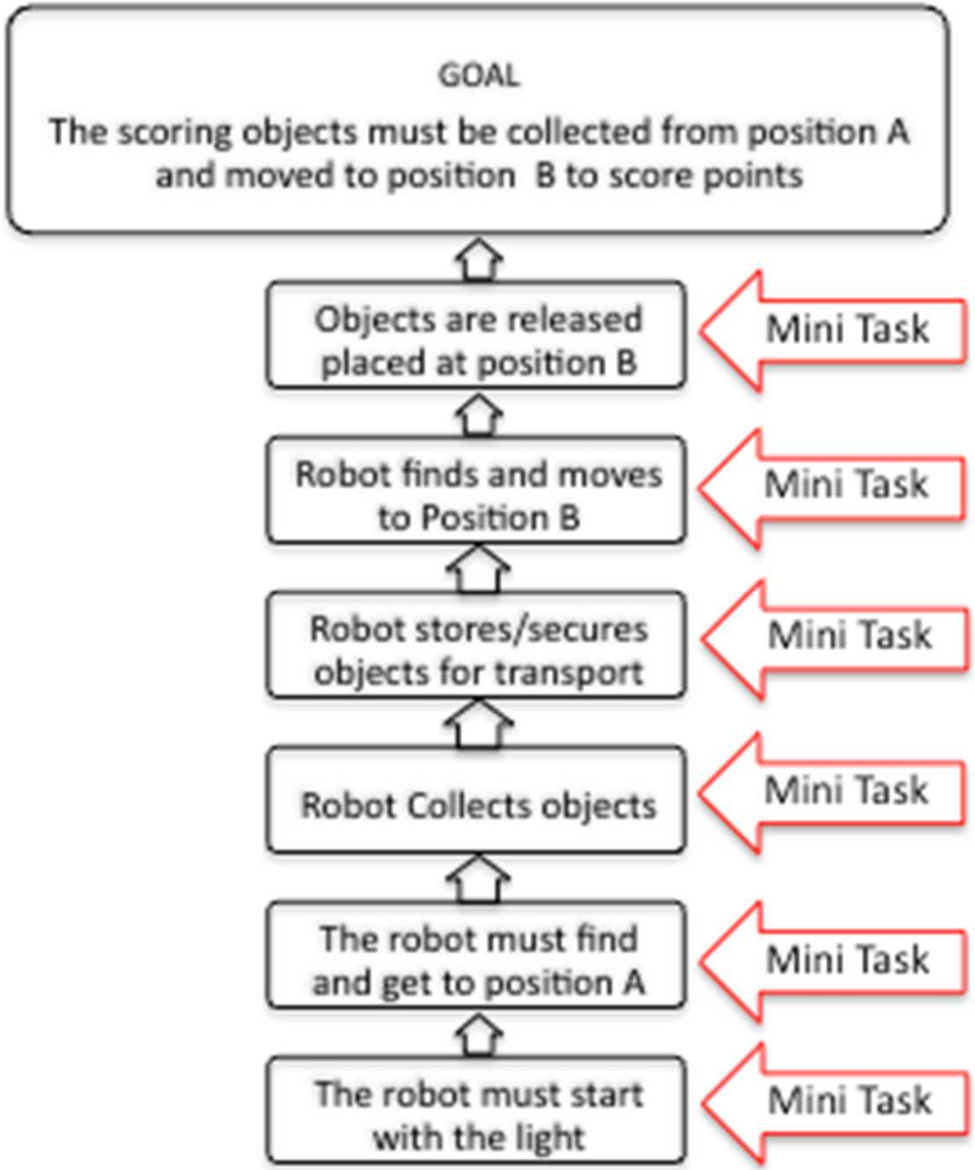
- Start with the **"easy points"** by having the students discuss and document **what has to happen** to score these points (goal) by working backwards (Task Analysis) from the desired goal.
- Have a planning strategy meeting to set common goals
- Working backwards helps the students focus on the goal and the step-by-step, sub task or "Mini Tasks" they have to accomplish to complete the final task.
- Keep calendar up to date with tasks and assignments

How do I teach this?



Botball®

How do I teach this?



- Move on to the more complicated “What has to happen” Harder
  Task A (harder points)

How do I teach this?

- When students want the robot to do (task A) and then (task B) and then (task C) the charts add together making the mini tasks needed to accomplish both summative and dependent upon one another.
- Finish task A before moving on to task B.
- After completing task B recheck the functionality of task A.
- **REMEMBER THE LAST TASK** is always to shut the robot down.
- Students will have no basis to predict how long it will take them to complete mini tasks (often underestimating the time required).
- In the end, if they are successful in completing only one task (goal) A, they have been successful and most likely will be competitive at the tournament

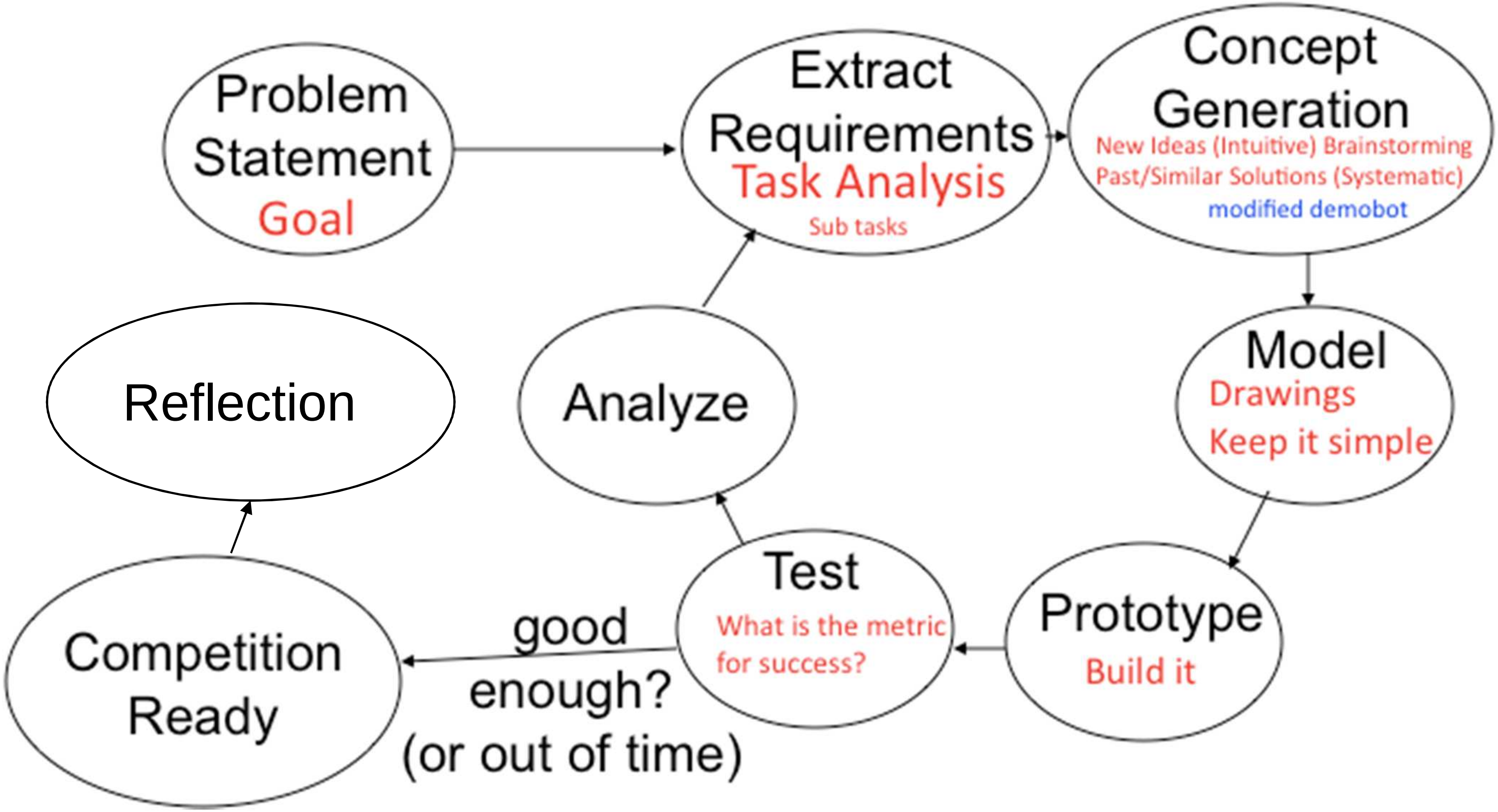
How do I teach this?

The students have arrived and asked; what do we do?

- Is their robot successfully and reliably completing mini task A or completing the wait for light and starting? Yes, great, now start working on mini task B.
- Many students will successfully complete the mini task once and think that is sufficient.
- Develop a metric to determine success, (It must wait for the light to start and work 10 out of 10 times or 4 out of 5 etc).
- If the task is continually giving them problems, reevaluate. Maybe a mechanical adjustment will make it work or have they double checked their code? Then explore solutions in the workshop examples, the Botball community site or calling KIPR?

How do I teach this?

The Engineering Life Cycle provides an overview of the entire process.



Tournament Awards

Tournament Awards

There are a lot of opportunities for teams to win awards

- Tournament Awards
  - Outstanding Documentation
  - Seeding Rounds
  - Double Elimination
  - Overall (Includes documentation+seeding+double elimination)
- Judges' Choice Awards (the number of awards is dependent on number of teams participating)
  - KISS Award
  - Spirit of Botball
  - Outstanding Engineering
  - Outstanding Software
  - Spirit
  - Outstanding Design/Strategy/Teamwork

When you come to the tournament.

- Adults are NOT allowed in the pits (you can help them get settled in and then you must leave)
- Bring ALL of your equipment. Especially your charging cables, extra LEGO etc.
- Bring your computers.
- Bring a power strip.
- Plan on staying for the awards (There are a lot of Judge’s Choice Awards)
- Check the Botball FAQ!
- Make a Checklist! (Tournament supplies and Robot setup.
- Prepare Your Onsite Presentation!
- Money for lunch or sack lunches.
- A CD or flash drive with a back up file of your code.

Preview of This Year's Game

Q&A is Next

BOPD

- What?
  - Botball Online Project Documentation
- When?
  - 3 periods during design and build portion
  - 1 onsite presentation at regional tournament
- Why?
  - To reinforce the engineering process
  - **POINTS EARNED IN BOPD FACTOR INTO OVERALL TOURNAMENT SCORE!**

See BOPD Handbook on the Team Home Base for more information

Botball T-Shirts

Team preorder - $7
At tournament - $10

Note:
T-shirts are not provided
One preorder per team
Can be modified up to 1 week prior to tournament

botball.org/shirts

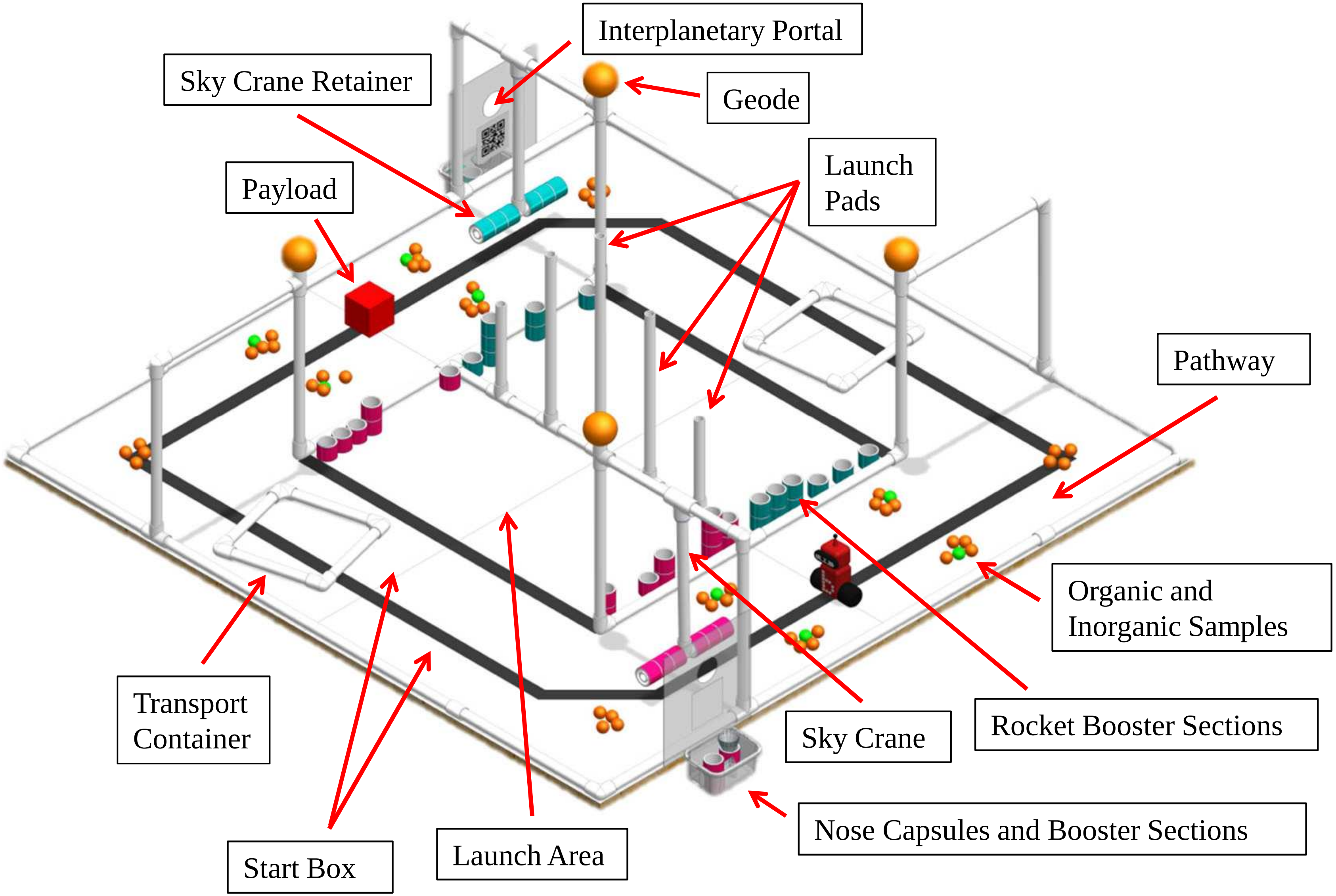
Tournament Awards

Tournament Awards

There are a lot of opportunities for teams to win awards

- Tournament Awards
  - Outstanding Documentation
  - Seeding Rounds
  - Double Elimination
  - Overall (Includes documentation+seeding+double elimination)
- Judges’ Choice Awards (the number of awards is dependent on number of teams participating)
  - KISS Award
  - Spirit of Botball
  - Outstanding Engineering
  - Outstanding Software
  - Spirit
  - Outstanding Design/Strategy/Teamwork

The Game Board



Game Q&A

Vision

Prep

Vision

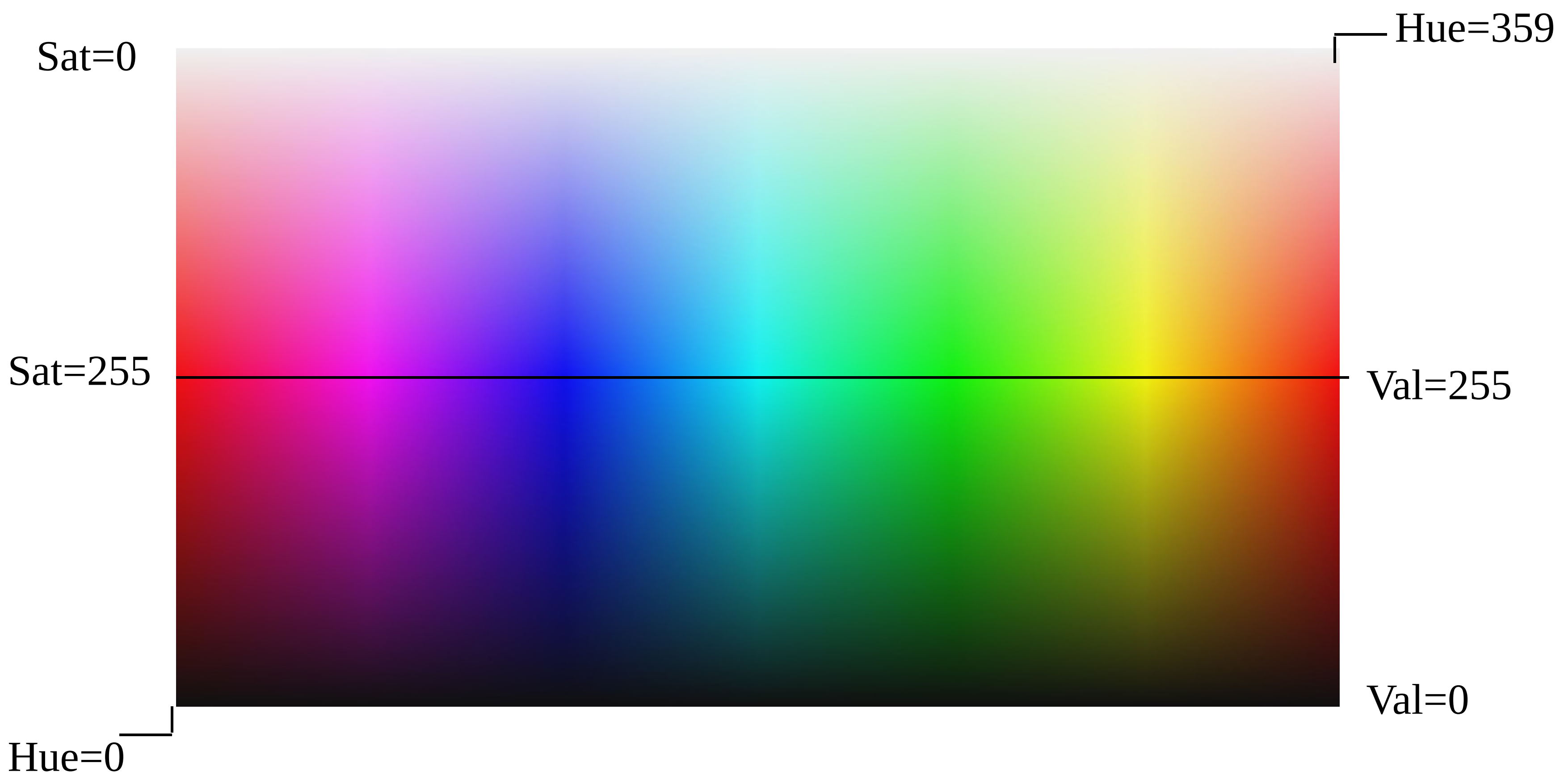
- Vision setup
- HSV color selection and color blobs
- Training the Link to use an HSV color model
- Using QR codes
- Library functions for using the camera

```
camera_open(<res>)
camera_close()
camera_update()
get_object_count(<ch>)
get_object_bbox(<ch>,<obj>)
get_object_center(<ch>,<obj>)
get_object_data(<ch>,<obj>)
camera_load_config(<name>.conf)
```

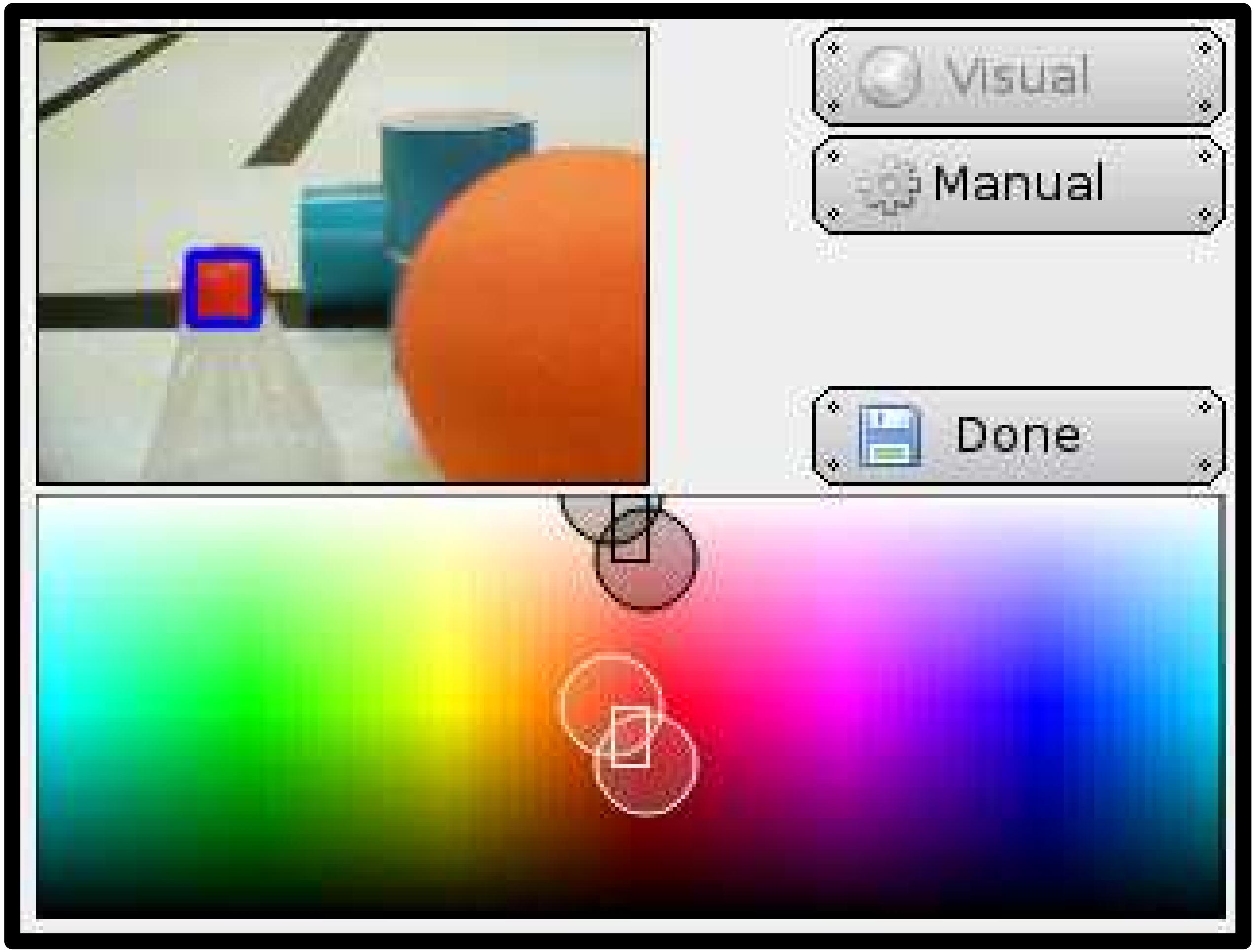
Vision Setup

- The USB cameras in the Botball kit will work in either of the Link's USB ports
- Plug in a camera and you will be able to see the camera image by going to the *Camera* screen under the *Motors and Sensors* menu
  - If you unplug the camera, the Link may no longer recognize it if you plug it back in
    - You will need to restart the Link if this happens

HSV Color Selection Plane



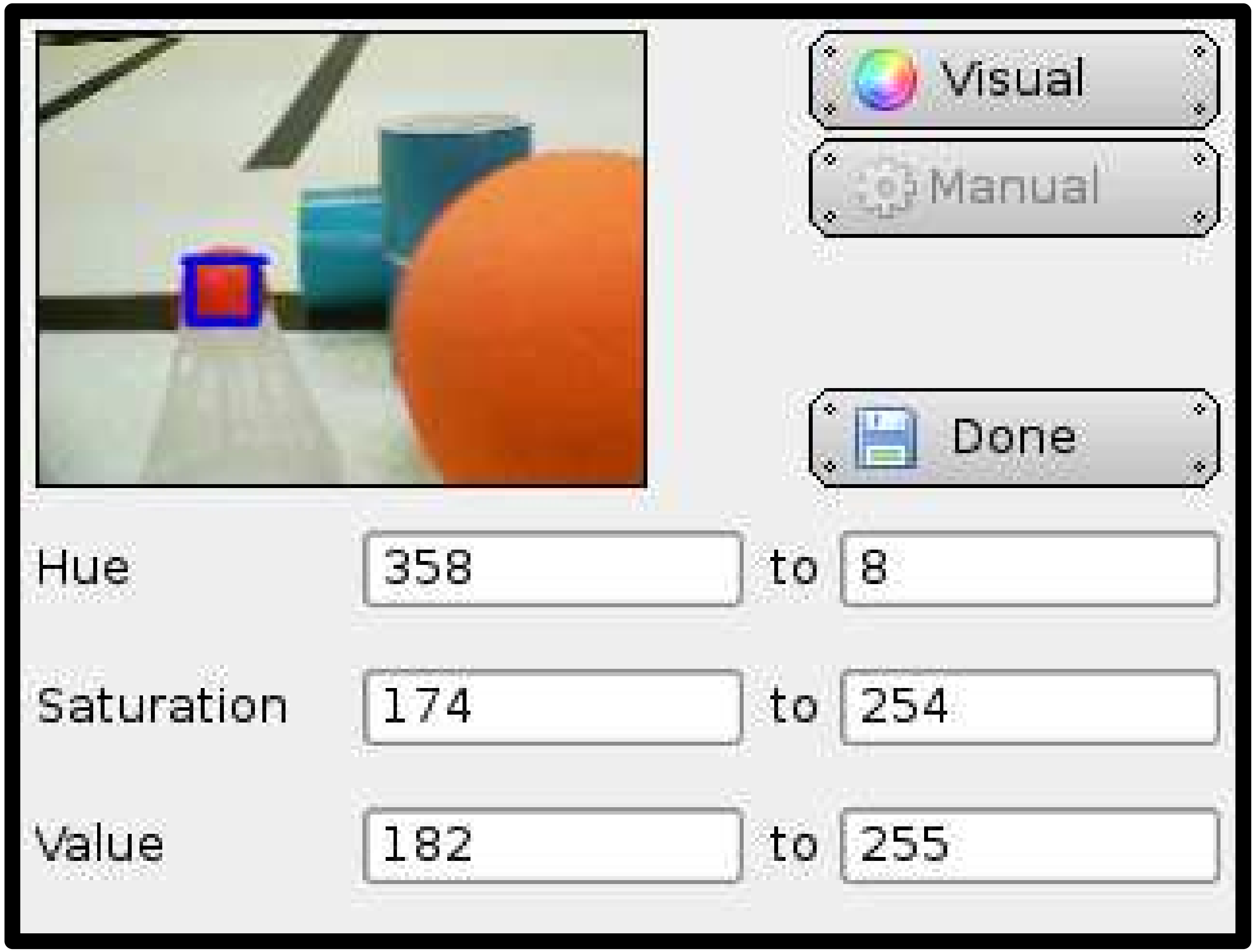
Channels Interface



Color Blobs

- Each pixel on the screen has an HSV color
- When we say "red", we really mean a range of HSV colors on the color selection plane that are approximately red
- Two rectangular pieces of the color selection plane that correspond to being "red" specify the range of HSV colors to be viewed as red by the KIPR Link
  - This is called an HSV color model
- A red *blob* is all contiguous pixels matching one of the HSV colors in the red range
- A blob has a bounding box, a center, etc.
- If you want to find Botguy with the camera, you look for a big red blob

Manual Channel Interface



Demo/Video of Setting Color Models

http://youtu.be/nSszFa7opMA

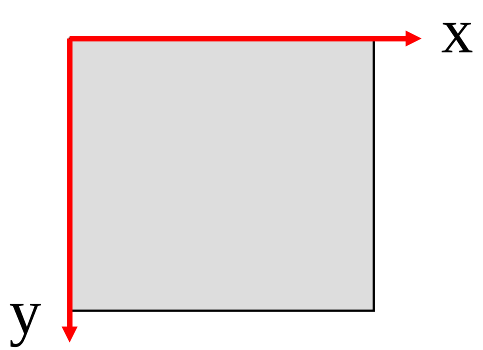
Performance Factors

- Focus:
  - A slightly blurred image smooths out colors and can improve some tracking reliability
  - Sharp focus is important for separating adjoining blobs and for QR codes
  - Adjust focus by turning the focus ring on the camera
- Image Resolution:
  - The lower the resolution the higher the frame rate
  - This is set by the argument given to camera_open(<res>)
    - HIGH_RES sets the image to 640x480
    - MED_RES sets the image to 320x240
    - LOW_RES sets the image to 160x120 - recommended

Image Coordinates

- The camera's processed field of view is treated as an x-y (column, row) coordinate array
  - The upper left corner has coordinates (0,0)
  - The lower right corner has coordinates (159,119) in `LOW_RES`
  - The Link display may distort the camera's field of view

What are the coordinates of the center?



Botball®

Vision System Color Models

- You can create multiple vision system configurations
  - Each configuration can have up to 4 channels
- **YOU MUST SET ONE CONFIGURATION AS THE DEFAULT**
- The KIPR Link can handle 4 Channels simultaneously
- Each channel can be either a HSV blob tracking channel or a QR code scanner channel
  - If you are tracking QR codes then you are limited to 3 simultaneous HSV blob channels
  - You never need more than one QR channel (since it can read all QR codes).

Vision System Library Functions
camera_update, **get_object_count**, **get_object_center**

- The KIPR Link library function **`camera_update();`** is a command that causes the KIPR Link to capture the most recent camera frame for analysis
  - Frame analysis determines objects properties such as the (x,y) coordinates of the center of the object
- **`get_object_count(3);`** provides how many objects are being seen by channel 3 **in the default configuration**
  - If the count is 0 there are no objects; if -1 the channel does not exist
  - Objects are numbered 0,1,2, … from largest to smallest
- **`get_object_center(3,0).x;`** for channel 3, object 0, returns the value of the center x coordinate of the largest object

More Object Functions

get_object_

```
center(<ch>,<obj>).x
center(<ch>,<obj>).y
bbox(<ch>,<obj>).ulx
bbox(<ch>,<obj>).uly
bbox(<ch>,<obj>).width
bbox(<ch>,<obj>).height
area(<ch>,<obj>)
```

Selecting the Action to Perform
if – **else**

- For **while**, an action is performed so long as the condition check is true
- In contrast, for **if** – **else** , one action is performed if the condition is true and another if it is false
- Example:

```
if (get_object_count(0) > 0)
    { printf("There's a red blob\n"); }
else
    { printf("Don't see a red blob\n"); }
```

- The **if** control structure is a special case of **if** – **else**

Example Using Vision Functions

```
// Set up a camera configuration that is calibrated so that it recognizes
// a red colored object for color channel 0 before running the program
// and make sure that configuration is the default
int main() {    // Start up the camera and specify the resolution
   int x, y, color=0;  // set up for color channel 0 (red)
   camera_open(LOW_RES);
   printf("Looking for red\nPress A when ready\n\n");
   printf("Press B button to quit\n");
   while (a_button() == 0); // wait for A button
   while (b_button() == 0){ // run till B button is pressed
        camera_update(); // process the most recent image
        if (get_object_count(color) > 0){
                //get x, y for the biggest blob the channel sees
                x = get_object_center(color,0).x;
                y = get_object_center(color,0).y;
                printf("Biggest blob at (%i,%i)\n",x,y);
        }
        else{
                printf("No color match in Frame\n");
        }
        msleep(200);  // give user time to read
   }
   printf("Program is done.\n");
   return 0;
}
```

Activity 8a: Description

Vision

- Plug the camera into your KIPR Link
- Calibrate your vision system so that color model 0 picks up nearby pink colored objects
- Copy the example program and download it to your KIPR Link
- Move a pink object around in front of the camera to get a feel for the boundaries of the camera's field of view
- Modify the program so it prints out whether the blob is to the right or to the left
- Change the color channel for the program and repeat using a different color

Activity 8a: Reflections

Vision

- How do lighting and shadows affect your color model?
  - Be sure to calibrate your camera under the same lighting in which it will be used
- How stable are blobs as you move an object or camera around while training?
- How close to the boundaries for X (0-159) and Y (0-119) could you get the centroid reading for your object?

QR Code data

- `get_object_data(<ch>,<obj>)`
  - returns a string pointer
  - returns -1 if the channel or object does not exist
- `get_object_data_length(<ch>,<obj>)`
  - returns an int the length of the string
- `get_object_data(<ch>,<obj>)[0]`
  - returns the first character of the data (e.g., 'P')

Activity 8b: Description

Vision

- Plug the camera into your KIPR Link
- Add a QR Code channel to your camera configuration (note the channel # -- probably channel 1)
- Copy the example program on the next page and download it to your KIPR Link
- Display a QR Code on your laptop or phone and point the Link camera at it
- Modify the printf for QR codes longer than one letter, try using the printf format %s to display the entire data field:

```
printf("QR code begins with %s\n", get_object_data(0,0));
```

Activity 8b: Code

```
// Prints the center coordinate of P QR codes otherwise prints first letter
int main(){ // Assumes that channel 0 of default config is QR code
  int x,y;
  char q;
  camera_open(LOW_RES); //start up camera
  while(a_button()==0){
    camera_update();// get a new image
    if(get_object_count(0)>0){// is there a QR code?
      if(get_object_data(0,0)[0]=='P'){// Is it a P
        x=get_object_center(0,0).x;
        y=get_object_center(0,0).y;
        printf("P found at %i,%i\n",x,y);
      }
      else{ // QR code is not P
        q=get_object_data(0,0)[0];
        printf("QR code begins with %c\n",q);
      }
    }
  }
  printf("Done\n");
  return(0);
}
```

QR Codes

- Visit the website: http://www.qrstuff.com
- Allows easy generation of QR codes

Activity 8c: Description

Vision Tracking

- Modify the line following program (Activity 7) to track the largest object on a vision channel
  - Code should be the same whether or not your channels is for a colored object or a QR Code
- Operating the camera at LOW-RES means that the threshold is when x is 80
- Improvements:
  - Divide the vision field into three regions where if the object's center is to the left, turn left, if it is to the right, turn right and if it is in the center region, go straight forward
  - If not object detected on that channel is in view stop and wait

Activity 8c: Solution

Vision Tracking

```
/* Move the robot towards the largest object on channel 0.
   Robots stops if no object is detected*/
int main(){
  int ch=0, leftmtr=0, rghtmtr=3;  // identify channel and motors
  int high=100,low=-10;      // set wheel powers for arc radius
  camera_open(LOW_RES);
  printf("Move towards object on channel 0\n");
  printf("Press B button when ready\n\nPress side button to stop\n");
  while(b_button()==0) {}    // wait for button press
  while(side_button()==0){   // stop if button is pressed
    if(get_object_count(ch)>0) { // if object is seen...
      if(get_object_center(ch,0).x<65) {// if object is on left...
        motor(leftmtr,low); motor(rghtmtr,high);   // arc left
      }
      else { if(get_object_center(ch,0).x>95) {// if object is on right...
                motor(rghtmtr,low); motor(leftmtr,high);   // arc right
             }
             else {motor(rghtmtr,high); motor(leftmtr,high);}//go straight
      }
    }
    else {ao();}
  }
  ao();  // stop because button pressed
  printf("done\n"); return 0;
}
```

Additional Activities

- **Activity 9**: Point Servo at Colored Object
- **Activity 10**: Bang-Bang Control
- **Activity 11**: Proportional Control
- **Activity 12**: Identify and Approach a Target
- **Activity 13**: Bang Bang Control with DemoBot Arm
- **Activity 14**: Proportional Control with DemoBot Arm
- **Activity 15**: Accelerometer for Bump Detect
- **Activity 16**: Music on the Create
- **Activity 17**: Reduce Heading Errors

From Now Until Lunch

- Work on any earlier activities you haven't finished
- Try out one or two of the additional activities listed on the previous slide (activity details are located at the end of today's presentation)

TAKE YOURSELF TO LUNCH

Remainder of the Day

- Continue working on Activities
- Take advantage of having experts in the room - ask them your technical questions
- Make contacts with other schools
  - Schedule joint practice sessions
  - Schedule mini tournaments
  - Set up joint fund raising activities
  - Etc.

Things for ALL Teams to Remember

1. Review "New for 2013" on your Team Home Base
2. Review the BOPD Manual (Botball Online Project Documentation)
3. 2013 Documentation expects:
   - Online (3 submissions) and Onsite Presentation. (You can't win without good documentation and a practiced presentation) A scored example is on your Team Home Base.
4. Side A and Side B for this year's game have different colors
   - Robots should be designed and programs written for running on either A or B sides (KIPR Software will determine what side you will run on)
5. Teams are allowed to use at most 1 camera on their entry
6. Use the manuals and "hints for new teams" on your Team Home Base

Suggestions for New teams

1. Read the "Hints for New Teams Manual" on your Team Home Base
2. Hit the ground running (don't wait to get started)
3. It is okay to ask for help - use the team home base forums, community site and KIPR
4. If possible, build a practice board (instructions are on your Team Home Base - this is a great parent/mentor/student activity)
5. Keep It Simple Students (start out with one task and do it well before adding tasks - a simple robot is easier to build, repair and program)
6. Don't forget the documentation - read and follow the rubrics
7. Check out the "construction hints" pictures of drive trains, effectors and sensor mounts on your Team Home Base to help generate ideas.
8. Use the DemoBots as a good starting point and modify them as you go.
9. HAVE FUN!

Wrap-up: Avoid Embarrassing Problems at the Tournament

- Test your robots **from start to end**:
  - Shield your starting light sensors
  - Go through the entire starting sequence
    - Calibrate your light sensor(s) to the starting light
    - Make sure the robots stop when they are supposed to
      - verify with a stop watch!
- Have a check list of what to bring
  - On-site documentation materials
  - Make backups of software
  - Power strip, laptop power supply, chargers for Create and KIPR Links
  - Bring backups of software

Check www.botball.org and your Team Home Base ***regularly***

Good Luck!

Additional Activities

- **Activity 9**: Point Servo at Colored Object
- **Activity 10**: Bang-Bang Control
- **Activity 11**: Proportional Control
- **Activity 12**: Identify and Approach a Target
- **Activity 13**: Bang Bang Control with DemoBot Arm
- **Activity 14**: Proportional Control with DemoBot Arm
- **Activity 15**: Accelerometer for Bump Detect
- **Activity 16**: Music on the Create
- **Activity 17**: Reduce Heading Errors

Point Servo at Colored Object

return

Activity 9 (Prep)

Point Servo at Colored Object

- Calibrate your vision system so channel 0 matches for a colored object
- Remember from the vision activities how the vision coordinate system works and download the program below
- Move the object around in front of the camera to get a feel for the boundaries of the camera's field of view

```
int main() { // Activity 9 prep
  int x,y;
  camera_open(LOW_RES);
  printf("Looking for blob\n\nPress side button to quit\n");
  while(!side_button()){ //run until side is pressed
    camera_update(); // process the most recent image
    if(get_object_count(1) > 0){ // any blobs of the trained color?
      x = get_object_center(1,0).x; y = get_object_center(1,0).y;
      // store x,y of biggest blob
      printf("Color Blob at (%i,%i)\n",x,y); msleep(200);
    }
    else{ printf("No Blob in Frame\n"); }
  }
  printf("Program is done.\n");
  return 0;
}
```

Activity 9

- Write a program to have the arm on DemoBot keep pointing at an object moved up and down in front of the camera (hint: use the y position of the blob as a factor in your position determination)
- The camera has a vertical angle of view of approximately 60 degrees, or 1/3 of the range of motion of a servo
  - Servo range is 180 degrees -> 0-2023 servo-tics
  - Y coordinate covers 60 degrees from 0-119 pixels
  - Each pixel represents about 6 servo-tics

Activity 9 Solution

```
#define ARMPORT 0
int main()
{
  int offset=662,yFactor=6, x, y,sPos;
  enable_servos();// servos powered on
  while(side_button() == 0) {
    camera_update(); // get most recent camera image and process it
    if(get_object_count(0) > 0) {  // there is a blob
      x = get_object_center(0,0).x//    x coordinate
      y = get_object_center(0,0).y//    and y
      display_printf(0,4,"Blob is at (%i,%i)\n",x,y);
      set_servo_position(ARMPORT,offset+y*6);// Assumes center position of servo…
      // is aligned with vertical center of camera
    }
    else {
      display_printf(0,4,"No colored object in sight\n");
    }
    msleep(200);      // don't rush print statement update
  }
  disable_servos(); // servos powered off
  printf("All done\n");
  return 0;
}
```

Bang-Bang Control

Bang-Bang Control

- Bang-bang control, as its name implies, is a control strategy that changes power to a new value without a transition such as first slowing down
  - The effect is like bumper car bouncing back and forth between two walls; i.e., you slam into reverse when you hit one wall (bang) and then slam into forward when you hit the other (bang), never slowing down to soften the blow
  - Activity 7, line following, used bang-bang with the robot turning either hard left or hard right. So did 8c

Activity 10 (Objectives)

Bang-Bang Control

Write a program that monitors the floating analog "ET" sensor (Day 1, slide ~163) to keep the robot a certain distance away from a moving obstacle using bang-bang control.

Run the program on the KIPR Link using the DemoBot

Activity 10 (Pseudocode)

Bang-Bang Control

1. Set analog port 0 to floating analog.
2. Loop until side button is pressed.
   a. If the floating analog rangefinder on port 0 has a reading indicating less than about 6" then back up
   b. Otherwise, drive forward
3. Stop the DemoBot when side button is pressed

Activity 10 (Solution)

Bang-Bang Control

```
// *********  Bang Bang Control
int main()
{
  int distVal = 600; // change this value to be sensor reading at 6 inches
  // Step 1: Set analog port 0 to floating analog
  set_analog_pullup(0,0); // disable pullup on port 0 so port is "floating"
  printf("Back up if obstacle too close\n   otherwise go forward\n");
  printf("Press A button to start\n\n");
  while (a_button() == 0);
  printf("Press side button to stop\n");
  // Step 2: Loop until side button is pressed
  while (side_button() == 0) {
    // Step 2a: If the floating analog rangefinder on port 0 reads
    //    greater than distVal, back up.
    if (analog10(0) > distVal) { motor(0,-50); mav(2,-50); }
    // Step 2b: Otherwise, drive ahead
    else { motor(0,50); motor(2,50); }
  }
  // Step 3: Stop
  ao();
  printf("done\n");
  return 0;
}
```

Activity 10 (Experiments)

Bang-Bang Control

- What is the behavior as you move an obstacle towards or away from the robot?
- Increase or decrease the distance the robot should stay from the wall. Do you notice a difference in sensor performance?

Activity 10 (Reflections)

Bang-Bang Control

- What else could bang-bang control be used for?
- Describe the behavior of the robot using bang-bang control.
- How can bang-bang control be improved
  - Think about activity 8c

Proportional Control

Proportional Control

- For bang-bang control, motion values change instantly when a target is reached.
- For proportional control, motion values are changed proportionally to the difference between an *actual* and a *desired* value
- Proportional control works best when controlling motors with velocity commands (e.g., **`mav`**) rather than power commands (e.g., **`motor`**)
- If we want the robot to maintain a distance equivalent to sensor value of 600 then we can set the robot's velocity to:

```
velocity = kP*(600 – analog10(0));
```

  - range sensor values > 600 mean that the robot is too close and generate negative velocities
  - range sensor values < 600 indicate For proportional control, motion values are changed proportionally to the difference between an actual and a desired value
  - **`kP`** can be set to adjust the responsiveness as desired

Activity 11 (Objectives)

Proportional Control

Write a program (or modify Activity 10) that monitors the "ET" sensor (Day 1, slide ~163) to keep the robot a certain distance away from a moving obstacle using proportional control.

Run the program on the DemoBot.

Activity 11 (Pseudocode)

Proportional Control

1. Set analog port 0 to floating analog.
2. Loop until side button is pressed.
   a. Set the velocity of both motors of the robot to a value proportional to the difference between the reading from the analog rangefinder on port 0 and its reading for about 6 inches.
3. Stop the DemoBot

Activity 11 (Solution)

Proportional Control

```
// *********  Proportional Control
int main() {
  int velocity, distVal=600;
  double kP=1.5; // adjust
  // Step 1: Set analog port 0 to floating analog
  set_analog_pullup(0,0);
  printf("Back up if obstacle too close\n   otherwise go forward\n");
  printf("Press A button to start\n\n");
  while (a_button() == 0);
  printf("Press side button to stop\n");
  // Step 2: Loop until side button is pressed
  while (side_button() == 0) {
    // Step 2a: move the motors proportional to the distance to the obstacle
    velocity = kP * (distVal - analog10(0));
    mav(0,velocity); mav(2,velocity);
  }
  // Step 3: Stop
  ao();
  printf("done\n");
  return 0;
}
```

Activity 11 (Experiments)

Proportional Control

- What is the robot's motor behavior as it gets close to the stopping point? Observe the motor lights.
  - How does it change if the value of **kP** is changed?
- Increase or decrease the distance the robot should stay from the obstacle. Do you notice a difference in sensor performance?
- What happens when you move the obstacle inside about 4 inches from the ET sensor?

Activity 11 (Reflections)

Proportional Control

- How does proportional control compare to bang-bang control?
- What else could proportional control be used for?
  - would it improve color object tracking?
  - How about line following?
- Describe the behavior of the robot using proportional control.
- How can proportional control be improved?

Approach a Specific QR Code

Activity 12 (Objectives)

Identify and approach a QR code

- Identify and approach a QR code that is a 'P' or a 'T' based on initial user input.

Activity 12 (Pseudocode)

Identify and approach a QR code

- Create two buttons labeled "P" and "T" for the user to press
- Identify all QR codes
- Use a for loop to step through each QR code in the channel
- Check each QR code for information
  - If it matches the user input
    - Approach target – see activity 8c and slide ~277
  - If no matches found, spin to search
- Stop with bump

return

Pseudocode for Approach to Target

1. Keep moving until close enough (while)
2. if target is to the left, realign to left
3. otherwise if target is to the right, realign to right
4. and otherwise move on toward the target

The for Loop

- The for loop is an alternative to while that can be clearer if you are trying to loop a specific number of times.
  - initialize i to 0, loop while i < takes on the index value of each object representing a QR code, and add 1 to i at the end of each iteration.
  - then check each object and if it is the QR code for 'P'…

```
int i;
...
for(i=0;i<get_object_count(0);i++){
  if(get_object_data(0,i)[0]=='P'){
    ...
  }
  ...
}
...
```

How Close Am I?

- A good way to use the camera to determine proximity to a target is to note:
  - If the camera is higher off the ground than the target, as your robot approaches the target the y coordinate of the target increases; if the camera is lower – vice versa
  - The y coordinate value relates to how close you are!
  - Other properties of objects in the camera's view change systematically with distance
  - Your program can access these properties (see the Manual for a list of vision functions)

Botball®

Activity 12 (Reflections)

Identify and approach a QR code

- If you use higher speeds how well does the algorithm perform?
- Rather than spin moves, suppose you continue to move forward but angle back toward center
  - performance should improve
  - but are there risks?

Bang-Bang Control and Arm

Objectives

Using a servo motor to operate an arm

- Write a function to operate the servo that raises or lowers the arm on DemoBot

Prep

Using a servo motor to operate an arm

- Prep
  - Using servo motors
    - Review Day 1 servo motor section
  - Bang-bang control
  - Arm function

Using Servos (Recap)

- The KIPR Link library functions for enabling (or disabling) all servo ports:
  - **`enable_servos();`** activates all servo ports
  - **`disable_servos();`** de-activates all servo ports
- **`set_servo_position(2,925);`** rotates servo 2 to position 925
  - You can preset a servo's position before enabling servos so it will immediately move to the position you want when you enable servos
  - Default position when servos are first enabled is 1024
- **`get_servo_position(2);`** provides the current position of servo 2
  - Works only when servos are enabled
- The KIPR Link *Servo Test* screen can be used to center a servo or determine what position values to use once the servo is installed on a robot

Servos and DemoBot

- There is one servo on DemoBot for raising or lowering its arm
- The *Servo Test* screen can be used for each servo to determine limit settings
  - Arm fully *up* and arm fully *down*
- Record these values for later use

Bang-Bang Control

- Bang-bang control is a control strategy that changes power to a new value without a transition such as first slowing down
  - With a moving robot, it is bang-bang control if you slam into reverse when you hit a wall (bang) going forward and then slam into forward when you hit a wall going backward (bang), never slowing down to soften the blow
  - Snapping an arm up or down is also a form of bang-bang control

Arm Function (Bang-Bang)

- Assuming servos have been enabled
  - To raise the arm (bang)

    set_servo_position(*<armport>***,***<raised-position>***);**
  - To lower the arm (bang)

    set_servo_position(*<armport>***,***<lowered-position>***);**
- Function prototype

```
void arm(int up_down);
```

- Function strategy
  - **if** the parameter **up_down** is 1 raise the arm
  - **else** lower the arm

Selecting the Action to Perform
if – else

- For **while**, an action is performed so long as the condition check is true
- In contrast, for **if – else** , one action is performed if the condition is true and another if it is false
- Example:

```
if (up_down != 0) // bang
    {set_servo_position(ARMPORT, UPOS);}
else              // bang
    {set_servo_position(ARMPORT, DPOS);}
```

C Preprocessor
#define

- Before your program is compiled it is first examined by the C preprocessor for preprocessing commands
- **#define**
  - Equates a meaningful name to repeatedly encountered text
    - `#define LMOTOR 0`
    - `#define GET_PC get_motor_position_counter`
  - The preprocessor will replace all occurrences of **LMOTOR** with **0** and **GET_PC** with **get_motor_position_counter**; for example,

```
if (GET_PC(LMOTOR) < 30) { . . .
```

is equivalent to

```
if (get_motor_position_counter(0) < 30) { . . .
```

Program Steps

Using a servo motor to operate an arm

- Create **#define** statements for using servos
  - Names specifying limits of servo travel,
    UPOS is <*up-position*>, **DPOS** is <*down-position*>
  - Names for arm function action
    UP is 1, **DOWN** is 0
  - Names to remember the ports for the arm servo
    ARMPORT is **0**
- Arm function prototype
  ```
  void arm(int up_down);
  ```
- In your **main** function
  - **enable_servos();**
  - Repeat several times: raise the arm, sleep a bit, lower it, sleep a bit
  - **disable_servos();**
- Arm function definition

Activity 13

Bang-bang control with Demobot arm

- Write a function with prototype

  ```
  void arm(int up_down);
  ```

  that uses bang-bang control to raise/lower the arm on DemoBot and uses `#define` to assign more meaningful names to constant values
- Start and stop with button presses
- Test your function using a program as outlined in the program steps
- Adjust your arm's **UPOS** and **DPOS** values as necessary to improve the accuracy of the arm's movement

Reflections

Bang-bang control with Demobot arm

- How would you attach and operate a claw attached to the arm?
- Would arm modifications be needed?
- When would it be better to use a control strategy that raised/lowered the arm gradually?

Activity 13 (Solution)

Bang-bang control with Demobot arm

```
// Using a servo motor to operate an arm using bang-bang control
#define DOWN 0      // arm is raised
#define UP 1        // arm is lowered
#define UPOS 200    // servo position arm raised
#define DPOS 1200   // servo position arm lowered
#define ARMPORT 0   // servo port for arm
void arm(int up_down);  // prototype for arm function
int main() {
  arm(UP);                        // initialize arm position as up
  enable_servos();                // start servos with arm up
  printf("Lower and raise arm until side button pressed\n");
  printf("Press A button to start\n\n");
  while(a_button() == 0);
  while(side_button() == 0) { // repeat until user presses side button
    msleep(2000); arm(DOWN);  // leave up for 2 seconds, then lower it
    msleep(2000); arm(UP);    // leave down for 2 seconds, then raise it
  }
  disable_servos();           // shut down servos
  printf("Done!\n");
  return 0;
}
void arm(int up_down) {
  if (up_down != 0) set_servo_position(ARMPORT, UPOS);
  else set_servo_position(ARMPORT, DPOS);
}
```

Proportional Control and Arm

Objectives

Using proportional control to operate an arm

- Write functions to operate the servo controlling the arm servo on DemoBot, one to raise the arm and one to lower the arm
  - Have each function slowly speed up the servo after it starts, then slow it down as the limit of arm travel is neared

Botball

Prep

Using proportional control to operate an arm

- Prep
  - Proportional control
  - Speeding up then slowing down proportionally
  - Function for raising an arm

Proportional Control

- For bang-bang control, motion values change instantly when a target is reached
- For proportional control, motion values are lowered proportionately as the target is neared
  - Assume the servo position for fully down is given by the #define names **DPOS** and for fully up by **UPOS**, where **UPOS < DPOS**
  - These are best determined using the *Servo Test* screen
- As an example, assuming **enable_servos();** here's a loop that slows down the servo as it moves closer to being fully raised

```
while (srvpos > (UPOS+5)) { // quit if close enough
  // reduce srvpos by a smaller amount each time
  srvpos = srvpos - sqrt(srvpos-UPOS);
  set_servo_position(ARMPORT,srvpos);
  msleep(100);  // give time to move
}
set_servo_position(ARMPORT,UPOS); // finish up
```

Speeding Up, Then Slowing Down

- For an arm, it is useful to gradually increase speed as the arm moves, then slow it back down as it nears its target position
  - The midpoint of arm travel is

    ```
    midpt = UPOS+(DPOS-UPOS)/2;
    ```

    or half way between **UPOS** and **DPOS**
  - When moving away from **UPOS**, **sqrt(srvpos-UPOS)** increases and **sqrt(DPOS-srvpos)** decreases

    ```
    if (srvpos < midpt) amt=5+sqrt(srvpos-UPOS);
    else amt = 5+sqrt(DPOS-srvpos);
    srvpos = srvpos + amt;  // amt to change srvpos
    ```

    - **amt** is higher towards the midpoint and lower toward the up and down positions (**UPOS** and **DPOS**)
    - 5 is added to **amt** to ensure when using amt to move the servo it is at least 5 from its current position

Function to Raise Arm

Proportional control

- From the current position of the servo, if less than halfway to being raised, speed up and once past halfway begin slowing down

```
void raise_arm() {
    int amt, srvpos=get_servo_position(ARMPORT);
    int midpt = UPOS +(DPOS-UPOS)/2;
    while (srvpos > (UPOS+5) ) {  // quit if close enough
        if (srvpos < midpt) amt = 5 + sqrt(srvpos-UPOS);
        else amt = 5 + sqrt(DPOS-srvpos);
        srvpos = srvpos - amt;    // move closer to UPOS
        set_servo_position(ARMPORT,srvpos); // move arm
        msleep(100);  // give time to move
    }
    set_servo_position(ARMPORT, UPOS); // finalize at UPOS
}
```

Activity 14

Using proportional control to operate an arm

- Write functions with prototypes

```
void raise_arm();
void lower_arm();
```

  that use proportional control to operate the arm on DemoBot, speeding up through the midpoint of travel then slowing down
  - Rework the **raise_arm** example to get **lower_arm**
- Test your functions by writing a **main** function to raise the arm and then lower the arm, repeating until the side button is pressed
  - Don't forget to put in **#define** names and values for **UPOS**, **DPOS**, and **ARMPORT**

Reflections

Using proportional control to operate an arm

- Are there any advantages for using non-linear scaling instead of linear scaling for proportional control?
  - What would be the effect of changing the constant 5 used in the examples to other values?
- You could raise and lower the arm using bang-bang control – how would this affect being able to hold something in a claw attached to the arm?
- How would you write a single function for raising/lower the arm (like done for bang-bang raising/lowering the arm), and make it independent of whether **`UPOS < CPOS`** or vice-versa?

Activity 14 (Solution)

Using proportional control to operate an arm

```
//  Using proportional control to operate an arm
#define UPOS 200    // servo positions for arm up
#define DPOS 1200   // servo positions for arm down
#define ARMPORT 0   // servo port for arm
void raise_arm();   // prototype for arm function
void lower_arm();   // prototype for arm function
int main() {
  set_servo_position(ARMPORT, DPOS); // initialize arm down
  enable_servos();                   //   and start servos
  printf("Lower and raise arm until side button pressed\n");
  printf("Press A button to start\n\n");
  while(a_button() == 0);
  while(side_button() == 0) {       // repeat until user presses button
    msleep(1000); raise_arm();      // leave down briefly, then raise it
    printf("Arm is up\n");
    msleep(1000); lower_arm();      // leave up briefly, then lower it
    printf("Arm is down\n");
  }
  disable_servos(); // shut down servos
  printf("DONE!");
  return 0;
}
```

(Continues on next page)

Botball®

return

Activity 14 (Solution, Cont'd)

Using proportional control to operate an arm

```
void raise_arm() {
   int amt, srvpos = get_servo_position(ARMPORT);
   int midpt = UPOS +(DPOS-UPOS)/2;
   while (srvpos > (UPOS+5) ) {  // quit if close enough
        if (srvpos < midpt) amt = 5 + sqrt(srvpos-UPOS);
        else amt = 5+sqrt(DPOS-srvpos);
        srvpos = srvpos - amt;    // move closer to UPOS
        set_servo_position(ARMPORT,srvpos); // move arm
        msleep(100);  // give time to move
   }
   set_servo_position(ARMPORT, UPOS); // finalize at UPOS
}
void lower_arm() {
   int amt, srvpos = get_servo_position(ARMPORT);
   int midpt = UPOS +(DPOS-UPOS)/2; // point of fastest travel
   while (srvpos < (DPOS-5)) {   // quit if close enough
        if(srvpos < midpt) {amt = 5 + sqrt(srvpos-UPOS); } // move amount
        else { amt = 5 + sqrt(DPOS-srvpos); }              // (at least 5)
        srvpos = srvpos + amt;   // move srvpos closer to DPOS
        set_servo_position(ARMPORT,srvpos); // move arm
        msleep(100);                    // give time to move
   }
   set_servo_position(ARMPORT, DPOS); // finalize at DPOS
}
```


Botball®

Accelerometer

Objectives

Detect a hit when backing up the Create module

- The Create has no rear bumper, so use the KIPR Link accelerometer to determine when it hits something while backing up

Prep

Detect a hit when backing up the Create module

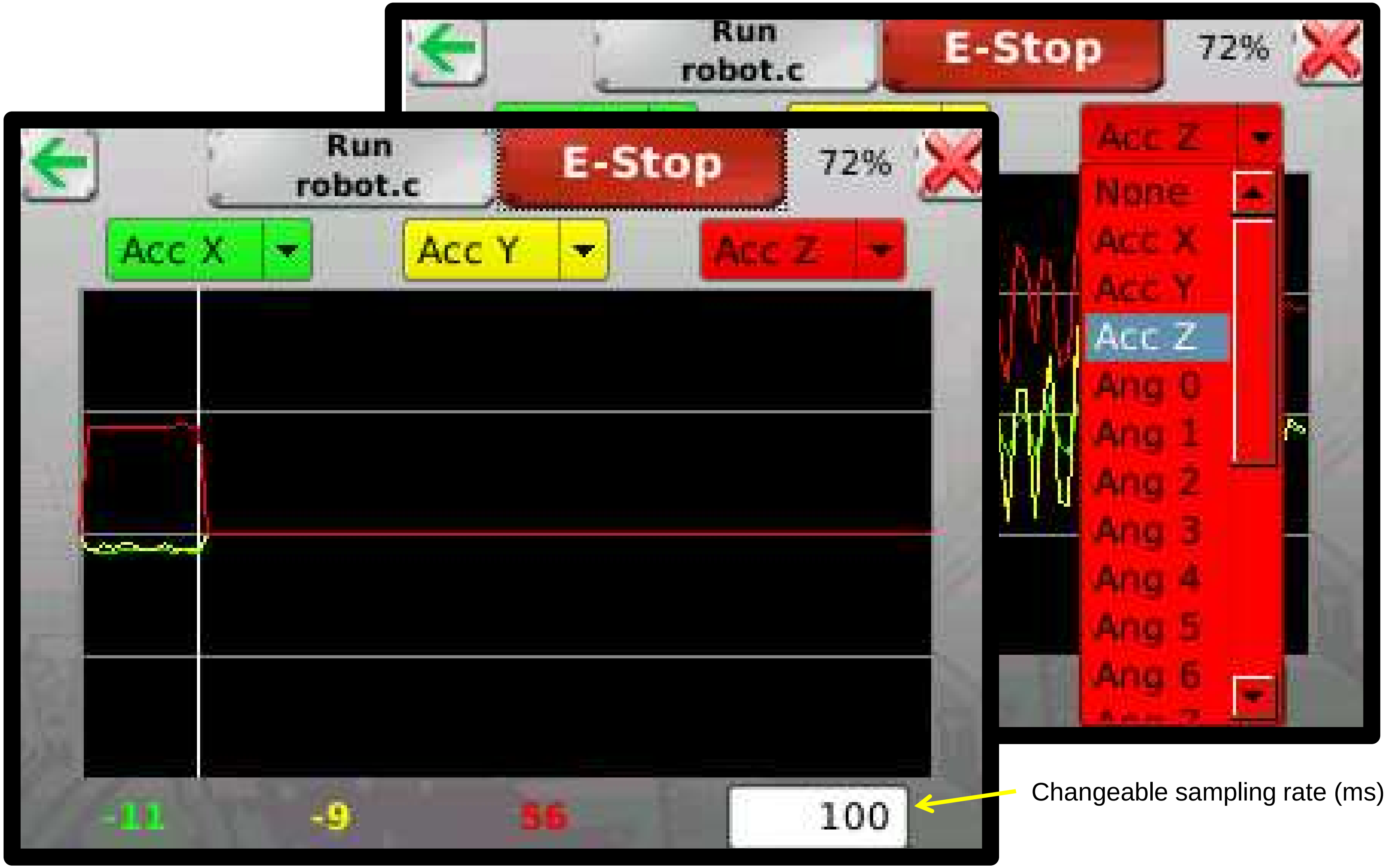
- Use the *Graph* screen to initially determine values the accelerometer generates when it is hit in the rear

Accelerometer

- An accelerometer measures force accelerating an object in 3 directions (vertical z, horizontal x, and horizontal y)
  - The y direction is front-to-back on the KIPR Link, x is left-to-right
  - Midpoint of 0, -512 to 511 range
- For an object at rest or moving on a flat surface at a constant speed the accelerometer measures no force for x and y
  - Gravity always exerts a force, so z > 0
  - Lining the KIPR Link up on the Create to look forward, it's rear is to the front of the Create and moving forward is the y direction
- The *Graph* screen shows this behavior for the KIPR Link's built in accelerometer (and can be used for other sensors as well)
  - scaling for the accelerometer has gravitational force (z value) at around 256 (perhaps off by 10-15%)
  - Suddenly stop the KIPR Link while moving forward to see y spike

return

KIPR Link Sensor *Graph* Screen



Changeable sampling rate (ms)

Sample **accel_x** Test

- Assume a variable **fwd_bk** is being used to determine if we're looking for a hit while going forwards (+1) or while going backwards (-1)
- For the case going backwards, **fwd_bk** is -1, so when positive acceleration (acceleration in the forward direction) is detected something has been hit

```
create_drive_straight(-250);
msleep(500); // get to constant speed
while (fwd_bk == -1) { // switch if hit
  if (accel_y() > 100) { fwd_bk = 1; }
}
```

Activity 15

Detect a hit when backing up the Create module

- Use the accelerometer to detect when the Create hits something while going forward or backward
- Pick the accelerometer axis (x, y, or z) that is aligned with the direction of motion
- When something is hit reverse direction
- Stop when the side button is pressed

Reflections

Detect a hit when backing up the Create module

- What happens if the accelerometer fails to detect a bump?
- What if you try to use different values (larger or smaller than 100) to detect a bump?

Activity 15 (Solution)

Detect a hit when backing up the Create module

```
// Using the accelerometer to detect a hit when moving the Create module
int main() {
  int fwd_bk = -1;
  int threshold = 100;
  create_connect();
  printf("Forward and back reversing direction on impact\n");
  printf("Press A button when ready\n\nPress side button to quit\n");
  while (a_button() == 0) {}
  while (side_button() == 0) {
    // monitor for a hit while going backward
    create_drive_straight(-250); // start going backwards
    msleep(500);                  // give time to reach constant velocity
    while (fwd_bk == -1) {
      printf("Moving backwards\n");
      if (accel_y() > threshold) { fwd_bk = 1; } // hit, reverse direction
      if (side_button() == 1) { break; }
    }
```

(Continues on next page)

Activity 15 (Solution, Cont'd)

Detect a hit when backing up the Create module

```
    // monitor for a hit while going forward
    create_drive_straight(250); // start going forwards
    msleep(500);                 // give time to reach constant velocity
    while (fwd_bk == 1) {
      printf("Moving forward\n");
      if (accel_y() < -threshold) { fwd_bk = -1; }
                                // hit detected, reverse direction
      if (side_button() == 1) { break; }
    }
  }
  create_disconnect();
  printf("done\n");
  return 0;
}
```

Music on the Create

Activity 16

- Look at the following example and Create your own music
  - You can't have more than 16 songs, 16 notes or fewer per song

Music on the Create

Useful Song Functions

- **`gc_song_array[16][33]`** can be used to store 16 songs.
- **`create_load_song(int`** *<song>***`)`** is used to load a song from **`gc_song_array`** into the Create.
- **`create_play_song(int`** *<song>***`)`** is used to load a song from **`gc_song_array`** into the Create.
- **`get_create_song_playing(int`** *<song>***`)`** returns **`1`** if a song is playing, **`0`** if it no song is currently playing.

Number	Note	Frequency	Number	Note	Frequency	Number	Note	Frequency
			60	C	261.6	96	C	2093.0
			61	C#	277.2	97	C#	2217.5
			62	D	293.7	98	D	2349.3
			63	D#	311.1	99	D#	2489.0
			64	E	329.6	100	E	2637.0
			65	F	349.2	101	F	2793.8
			66	F#	370.0	102	F#	2960.0
31	G	49.0	67	G	392.0	103	G	3136.0
32	G#	51.0	68	G#	415.3	104	G#	3322.4
33	A	55.0	69	A	440.0	105	A	3520.0
34	A#	58.3	70	A#	466.2	106	A#	3729.3
35	B	61.7	71	B	493.9	107	B	3951.1
36	C	65.4	72	C	523.3	108	C	4186.0
37	C#	69.3	73	C#	554.4	109	C#	4434.9
38	D	73.4	74	D	587.3	110	D	4698.6
39	D#	77.8	75	D#	622.3	111	D#	4978.0
40	E	82.4	76	E	659.3	112	E	5274.0
41	F	87.3	77	F	698.5	113	F	5587.7
42	F#	92.5	78	F#	740.0	114	F#	5919.9
43	G	98.0	79	G	784.0	115	G	6271.9
44	G#	103.8	80	G#	830.6	116	G#	6644.9
45	A	110.0	81	A	880.0	117	A	7040.0
46	A#	116.5	82	A#	932.3	118	A#	7458.6
47	B	123.5	83	B	987.8	119	B	7902.1
48	C	130.8	84	C	1046.5	120	C	8372.0
49	C#	138.6	85	C#	1108.7	121	C#	8869.8
50	D	146.8	86	D	1174.7	122	D	9397.3
51	D#	155.6	87	D#	1244.5	123	D#	9956.1
52	E	164.8	88	E	1318.5	124	E	10548.1
53	F	174.6	89	F	1396.9	125	F	11175.3
54	F#	185.0	90	F#	1480.0	126	F#	11839.8
55	G	196.0	91	G	1568.0	127	G	12543.9
56	G#	207.7	92	G#	1661.2			
57	A	220.0	93	A	1760.0			
58	A#	233.1	94	A#	1864.7			
59	B	246.9	95	B	1975.5			

Create Music Demo

- Dueling Banjos: E F G E F D E C D
- Create notes: 64 65 67 64 65 62 64 60 62

```
int main(){
     create_connect();
     printf("This program plays a song on the Create\n");
     gc_song_array[0][0]=9; //there are 9 notes in the song
     gc_song_array[0][1]=64;//E
     gc_song_array[0][2]=16;//.25 sec
     gc_song_array[0][3]=65;//F
     gc_song_array[0][4]=16;//.25 sec
     gc_song_array[0][5]=67;//G
     gc_song_array[0][6]=32;//.5 sec
     gc_song_array[0][7]=64;//E
     gc_song_array[0][8]=32;//.5 sec
     gc_song_array[0][9]=65;//F
     gc_song_array[0][10]=32;//.5 sec
     gc_song_array[0][11]=62;//D
     gc_song_array[0][12]=32;//.5 sec
     gc_song_array[0][13]=64;//E
     gc_song_array[0][14]=32;//.5 sec
     gc_song_array[0][15]=60;//C
     gc_song_array[0][16]=32;//.5 sec
     gc_song_array[0][17]=62;//D
     gc_song_array[0][18]=32;//.5 sec
     create_load_song(0);
     printf("Song is starting\n");
     create_play_song(0);
     printf("Song playing %d\n",get_create_song_number(.05));
     while(get_create_song_playing(.1)){}   //wait until song
     printf("Song has finished playing\n"); // finishes
     return 0;
}
```

Remove Accumulated Heading Errors

Activity 17

Remove Accumulated Heading Errors

- As your robot moves about its heading will drift from the robot's "idea" of its heading
  - When moving straight, the robot's heading will drift
  - When making a turn, the actual angle will differ from the angle specified
- You can have your robot occasionally do a maneuver to remove these accumulated errors using such methods as:
  - Have the robot follow a line of known heading
  - Have the robot physically align itself with a known object – which is the method we will explore in this activity

Physically Aligning Robot With a Wall

- Extended surfaces such as the border of the Botball field are ideal for reducing rotation errors
- Several methods can be used including:
  - Having a large flat front or rear of the robot and driving the robot so that the flat surface of the robot is forced against the wall. The disadvantage of this method is that there is no verification it has succeeded
  - Use two contact sensors on the front or rear edge of the robot (whichever end is going to be against the wall). When the first sensor makes contact with the wall, rotate the robot such that the second sensor contacts the wall while maintaining contact with the first sensor
  - Use the Create bumper against flat surface: there are two switches, if you find out range where both are pressed, and then move halfway through that range, Create should be aligned (assuming bumper activation is symmetric).

Activity 17
Using Two Sensors to Align to Wall

- Build a two bumper sensor (an example is on the left)
- Make sure the sensors can activate independently
- Attach it to one of your robots
- Write a program so that if the robot runs into a wall with the bumper it rotates until both sensors are activated

Aligning Robot With a Wall Using Create Bumper

```
//Define the base robot speed and the forward offset to keep the bumper pressed
#define SPEED 50
#define FOR 15
double timeTurn2Bumps(int dir);// function measures how long both bumpers are pressed
void turnHalfTime2Bumps(int dir, double time);//turn in direction dir for half of time
int main()
  {
    int dir=1;//dir = 1 means CCW and -1 means CW
    create_connect();  // connect to Create
    printf("press A to start towards a wall\n");
    while(a_button()==0);// wait for the A button to be pressed
    while(!get_create_lbump() && !get_create_rbump()){
      create_drive_straight(2*SPEED);// drive forward till either bumper is pressed
    }//if the left bumper and not the right is pressed, reverse rotation direction
    if(get_create_lbump() && !get_create_rbump()) dir=-dir;
    else {//if both bumpers are pressed, spin until only the right is pressed
      if(get_create_lbump() && get_create_rbump()){
        while(get_create_rbump()){
          create_spin_CCW(50);
        }
        msleep(100);//and then spin 0.1 sec longer
      }
    }//find how long, during rotation both bumpers are pressed
    turnHalfTime2Bumps(-dir,timeTurn2Bumps(dir)); //then turn back half that time
    printf("Create is orthogonal to wall. Done.\n");
}
```

(Continues on next page)

Aligning Robot With a Wall Using Create Bumper (Cont'd)

```
double timeTurn2Bumps(int dir){// dir determines which direction to rotate
  double startTime, turnTime;
  create_drive_direct(-dir*SPEED+FOR,dir*SPEED+FOR);//spin while pressing forward
  while(!(get_create_lbump() && get_create_rbump()));//wait till both bumpers are hit
  startTime=seconds();//seconds uses system clock
  while(get_create_lbump() && get_create_rbump()); //wait till both bumpers are hit
  turnTime=seconds()-startTime;//turnTime holds the amount of time both bumpers were pressed
  msleep(100);//turn a little further
  create_stop();//stop
  return (turnTime);//send back how long both bumpers were pressed
}

void turnHalfTime2Bumps(int dir, double time){
  double startTime;
  create_drive_direct(-dir*SPEED+FOR,dir*SPEED+FOR);//spin while pressing forward
  while(!(get_create_lbump() && get_create_rbump())); //wait till both bumpers are hit
  startTime=seconds();//startTime occurs when both bumpers are pressed
  while(seconds()-startTime < time/2.0);//keep turning for half of time
  create_stop();//stop and Create should be orthogonal against wall
}
```

END