# KISS Institute for Practical Robotics
# Botball Kit Documentation

This documentation was written by David Miller, Randy Sargent, Charles Winton, Holly Yanco, Illah Nourbash, JJ Maybury & Terry Grant






## Caveats & Warnings

- This is general purpose documentation
- Your robot kit may not contain all of the parts mentioned in the documentation
- This documentation refers to the Handy Boards distributed by KISS Institute in 2001 and 2002. Other boards may have different port assignments

# Overview of Documentation

- Download and install IC
- Overview of basic IC programming
- Overview of Handy Board
  - Handy Board sensors
- Overview of Motors
- Overview of RCX and sensors
- Example LEGO robot building instructions





# Interactive C
# New & Improved for 2002

- Interactive C 4.0 is a C compiler/interpreter
- Implements most of the ANSI C language
- Runs on Handy Board and LEGO RCX
- Loads firmware ("pcode" interpreter) onto board
- Provides an editor and documentation
- Provides an interactive environment for testing and debugging

# Getting IC 4

- The software 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 consider KISS Institute
  - If you would like to use IC 4 in a commercial product, contact KISS Institute about licensing
- The latest version of IC may be downloaded from: www.kipr.org/ic





# Setting Up

- IC runs partly on the PC and partly on the robot board
- For IC to fully operate, a Handy Board or RCX needs to be connected
  - The robot board needs to have the firmware (pcode interpreter) loaded in order to be able to talk to IC
- The IC editor can be used without an attached robot board

# To Install IC

- On a Mac OSX
  - Double click on IC4.tgz file
    - Note: keep the app and the library folders in the same folder
- On a Mac OS 8 & 9
  - Double click on IC4.sit file
    - Note: keep the app and the library folders in the same folder
- On Windows
  - Double click on IC4-install.exe
    - IC4 will be added to your program menu
    - Be sure to uninstall earlier versions before installing new one
- On Linux
  - Extract from tar archive on CD to your directory
    - Note: keep the app and the library directories in the same directory





# Get Started with IC Environment

- Start the IC application
- Click on the picture of the Handy Board
- Click on the **Connect Later** button
- Click on the **New** button (upper left corner)
- Type in the program that prints out a text string
- Save the file using the **Save** button (name it whatever you want -- just remember where you put it).

# Download the Firmware

- Make sure your Handy Board is connected to your personal computer via serial port cable, serial interface box and 4-pin modular cable
- Select **Download Firmware** from the ***Settings*** menu
- Select the appropriate serial port
- Click on **Download Firmware** button
- Follow the onscreen directions





# Interacting with IC

- Click on the **Interaction** tab
- Just type into area at bottom of IC window
- Simple expressions

```
2+2;
```

- Making noise

```
beep();
```

- Printing to the LCD screen

```
printf("I'm printing!!\n");
```

# Download a Program

- Select the tab with your program's name and click download
- To run your program (**`main`** function), turn the Handy Board off and then turn it back on
- To turn your HB on without running your program, hold down the start button while sliding the switch





# Data Types

- **`int`** (16 bit integer number in IC)
  +/-32,767
- **`long`** (32 bit long integer number)
  +/-2,147,483,647
- **`float`** (32 bit floating point number)
  e.g. 3.1416
- and others (see IC documentation)

# Integer Arithmetic

- + addition
  X + Y means X plus Y
- - subtraction
  X - Y means X minus Y
- * multiplication
  X * Y means X multiplied by Y
- / division
  X / Y means X divided by Y with the decimal truncated
- % modulus (remainder)
  X % Y means the remainder of X divided by Y





# Floating Point Arithmetic

- +,-,* operate the same as with integers
- / yields a floating point number
- % is not defined with floating point numbers

# IC Interaction Environment

- You can type C statements (and C blocks) in the interaction window
- Pressing return will send that code to the IC interpreter which executes the code on the Handy Board
- This is a quick way to test things
- Try it!





# Useful Function: `sleep(`*seconds*`);`

- `sleep()` delays the function's execution for an amount of time equal to the number of seconds (expressed as a float) given as an argument

# C Function: `printf();`

- Takes a quoted string and prints it out
  - %d is used for an integer
  - %f is used for a float
  - \n starts next character back at upper left corner (or new line on terminal)
  - commas separate all arguments after the quoted string





# More IC Library Functions

- `start_button();`
- `stop_button();`
- `knob();`
- `beep();`

# New Features of IC 4.0

- Tools menu
  - List functions
  - List global variables (e.g., **`servo0`**)
  - List loaded files
- Settings
  - Huge font
  - Download firmware





# Recap: Data Types

- **`int`** (16 bit integer number in IC)
  e.g. +/-32,767
- **`float`** (32 bit floating point number)
  e.g. 3.1416
- and others (see IC documentation)

# Variables

- Retain data for later use
- Use in constructions such as arithmetic expressions
- Must be declared before use at start of function (***local variables***) or outside any function (***global variables***)
  - Syntax: <*data-type*> <*variable-name*>
  - `int i;` [*integer variable*]
  - `float x;` [*floating point variable*]
- To put a value in a variable:
  - Syntax: <*variable-name*> = <*expression*>
  - `i = 2;`
  - `x = 2.0 * 1.5;`
  - `x = 3.2 * (float)i;`





# Program Flow

- When a program is run, the control moves from one statement to the next
- **Be careful of `1` vs `l`** (one vs ell)
- Calculate $j^2 + 1$ when j = 3

```
void main()
{
   int r,j;        /* declare r and j */
   j = 3;          /* assign a value to j */
   r = j*j + 1;    /* calculate and assign */
   printf("result is %d\n",r);
}
```

# Repeating Execution: Loops

- Loops are used when you want to do the same thing multiple times
- E.g., beep 20 times
- You could type `beep();` 20 times, but there is a better way: a loop





# Repetition Using `while`

- Syntax: `while` (<*test*>) {*statements*}
- Beep 20 times

```
void main()
{
   int num;           /* declare counter */
   num = 1;           /* initialize counter */
   while (num <= 20) /* loop while num is <=20*/
   {
      beep();         /* beep once */
      num = num + 1; /* add one to the counter */
   }
}
```

# IC: `while` (<sensor-test>)

- Syntax recap: `while` (*<test>*) {*statements*}

```
void main()
{
   int j=0; /* to tell how many times around the loop */
   while(stop_button()==0)
   {
      j=j+1;          /* next time around */
      printf("loop %d - press stop to stop\n",j);
      fd(0);          /* turn on motor 0*/
      fd(2);          /* turn on motor 2 */
      sleep(1.0);     /* wait for 1 second */
      bk(0); bk(2); /* reverse both motors */
      sleep(1.0);     /* wait for 1 second */
   }
   ao();  /* turn off all the motors */
}
```





# Boolean Expressions

- Boolean expressions result in either *1 (true)* or *0 (false)*
- Boolean operators:
  - == (two equals signs together, not one)
  - <, <=, >, >=
  - != (not equal)
  - || (or), && (and)
  - ! not
- Already seen a boolean expression in while:
  - `while` (*<boolean expression>*)

# Making a Decision with `if`

- Syntax `if` (<*test*>) {*statements*}
- The statements are skipped if the test is false

```
void main()
{
   int j=-2;
   if (j < 0)          /* skip if >= 0 */
   {
      j = -j;          /* change sign of x */
   }
   printf("magnitude is %d\n",j);
}
```





# Either/Or Selection: `if-else`

- Syntax `if` (<*test*>) {*statements for true case*} `else` {*statements for false case*}
- If the test is true {*statements for true case*} are selected
- If the test is false {*statements for false case*} are selected

```
void main()
{
   int j=-2;
   if (j < 0) /* select less than 0 case */
   {
      printf("%d is negative\n",j);
   }
   else       /* select greater than or equal to 0 case */
   {
      printf("%d is non-negative\n",j);
   }
}
```

# IC: `if` (<*sensor-test*>) … `else`

- Syntax recap: `if` (<*test*>) { …} `else` { …}

```
void main()
{
   if(digital(15)==1)     /* select "forward" case */
   {
      motor(3,50);
      printf("Forward\n");
   }
   else                   /* select "reverse" case */
   {
      motor(3,-75);
      printf("Reverse\n");
   }
   sleep(2.0);            /* let motor run 2 secs */
   off(3);                /* stop motor */
}
```





# IC: Selection Nested in Loop

```
void main()
{
   while (stop_button()==0)    /* loop until stop */
   {
      if(digital(15)==1)       /* select "forward" case */
      {
         motor(3,50);
         printf("Forward\n");
      }
      else                     /* select "reverse" case */
      {
         motor(3,-75);
         printf("Reverse\n");
      }
      sleep(2.0);              /* let motor run for 2 secs */
   }
   off(3);                     /* turn off motor */
}
```

***Teaser: what happens if you press and release stop while it is sleeping?***

# IC: Functions & Processes

- Functions are called sequentially
- Processes can be run simultaneously

  `start_process(`*function-call*`);`
  - processes halt when function exits or parent process exits
  - processes can be halted by using

  `kill_process(`*process_id*`);`
- `hog_processor();` locks process in CPU until it finishes or defers
- `defer();` causes process to give up the rest of its time slice until next time.





# Example of Processes

```
void main()
{
   int pid;
   /* run my waa function    */
   /* as a parallel process */
   pid=start_process(waa());
   sleep(5.0);
   /* if waa is still going */
   /* turn waa off          */
   kill_process(pid);
   /* if tone got killed    */
   /* finish it off         */
   beeper_off();
}
```

```
void waa()
{
   float p;
   while(!stop_button())
   {  /* run tone up */
      /* over & over */
      p=200.0;
      while(p<5000.0)
      {
         tone(p,0.05);
         p=p+200.0;
      }
   }
}
```

# Example of Processes (2)

```
void main()
{
   int pid;
   // run s_waa function
   // as a parallel process
   pid=start_process(s_waa());
   sleep(10.0);
   /* if s_waa is still going
      turn s_waa off          */
   kill_process(pid);
   /* if s_waa timed out, turn
      off sound & servo*/
   beeper_off();
   init_expbd_servos(0);
}
```

```
void s_waa()
{
   float p;
   int s3; // servo value
   init_expbd_servos(1);
   while(!stop_button())
   {// run tone and servo up
      p=200.0; s3=200;
      while(p<3800.0)
      {
         servo3=s3;
         tone(p,0.1);
         p=p+200.0;
         s3=(int) p;
      }
   }
   init_expbd_servos(0);
}
```





# Loading Multiple Files

- **#use "filename.ic"**
  - The text above, if at the beginning of your file will download the file filename as well when your file is downloaded.
  - You can have multiple #use lines in your file
  - The files you #use can #use files of their own
  - #use will automatically not get caught in loops
  - The files you #use must either be in the same directory as the original file or in the library folder

# IC Programming

- The previous slides were just hilights
  - IC handles most of ANSI C language including multi dimensional arrays, structs, #define, for, etc.
  - See a C text for more details of the C language
  - Use the Tools menu to see a list of all of the C functions in the library

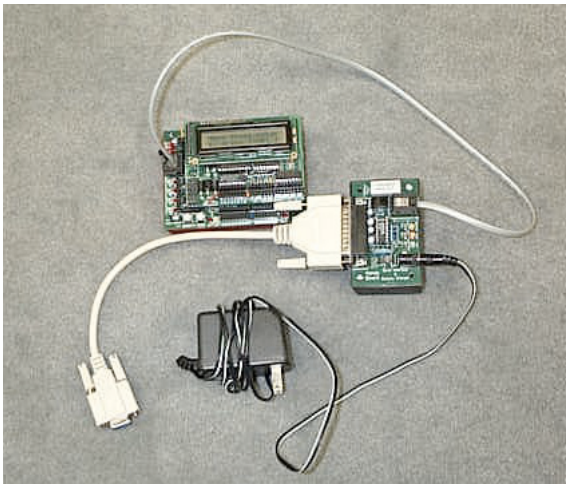




# Handy Board Checklist

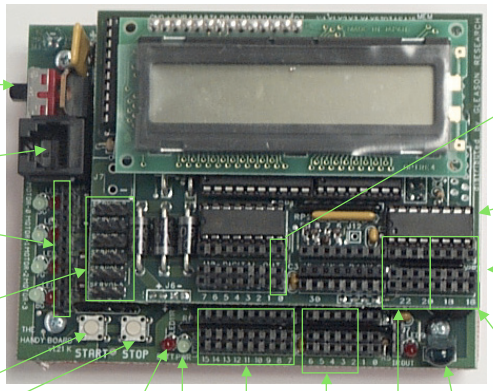
- Handy Board
  - Display
  - Expansion Board
  - Main Board
  - Battery Box
- HB to Interface cable (phone cord)
- Interface Board
- Computer to Interface cable (25 pin to 9 pin)
- AC Adapter (note: ground center, +12 outside)

# Handy Board Setup






# The Handy Board



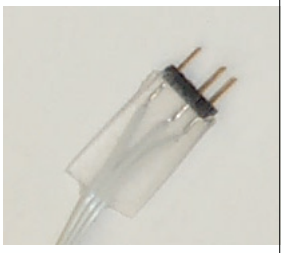
# Sensor Types

- Digital:
  - Return a 0 or a 1
  - Switches or bumpers are an example (open: 0, or closed: 1)
- Analog:
  - Sensor returns a continuum of values
  - Processor digitizes results (8 bits give values of 0-255)
  - e.g. light sensors




# Handy Board Sensors

- Knob
- Start and Stop Buttons
- light sensors: analog
- IR reflectance sensors: analog
- Optical rangefinder: floating analog
- assorted touch sensors: digital
- Slotted encoder sensors: digital
- Sonar rangefinder: (special ports)
- All detachable Handy Board sensors have keyed connector



# Built-in Sensors: Knob

- Knob: the knob is a potentiometer which is treated like an analog sensor
  - The access function `knob()` returns 0-255
  - Can be used for adjusting values during runtime





# Built-in Sensors: Start

- Start Button:
  - Access function `start_button()` returns 1 while pressed, 0 otherwise
  - Holding down Start while powering the board will bypass executing main (e.g., if your program immediately causes the robot to move, and you want to download new code, this is very useful)

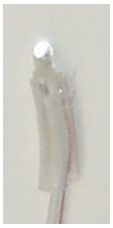
# Built-in Sensors: Stop

- Stop Button:
  - Access function `stop_button()` returns 1 while pressed, 0 otherwise
  - Holding down Stop while powering the board will put the board in download mode. This is necessary for loading the firmware.





# Light Sensors



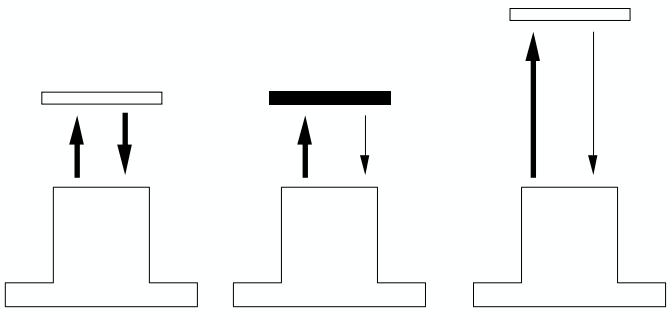
- Analog sensor
- Connect to ports 2-6 or 20-23
- Access with function `analog(`*port#*`)`
- Low values indicate bright light
- High values indicate low light
- Sensor is somewhat directional and can be made more so using black paper or tape or an opaque straw or lego to shade extraneous light. Sensor can be attenuated by placing paper in front.

# IR Reflectance Sensor "Top Hat"

- Analog sensor
- Connect to ports 2-6 or 20-23
- Access with function `analog(`*port#*`)`
- Low values indicate bright light, light color, or close proximity
- High values indicate low light, dark color, or distance of several inches
- Sensor has a reflectance range of about 3 inches




# IR Reflectance Sensors

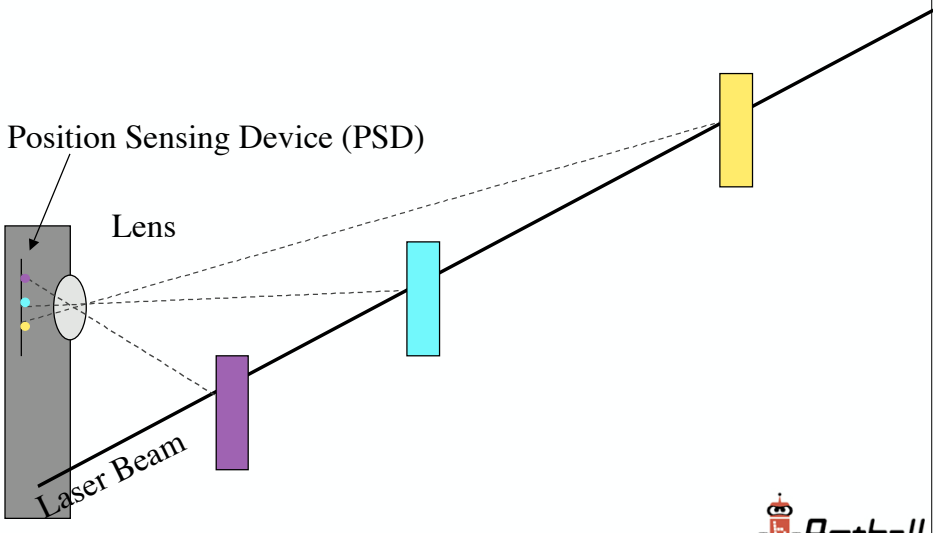


# Optical Rangefinder "ET"

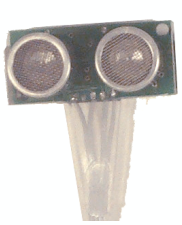
- Floating analog sensor
- Connect to ports 16-19
- Access with function `analog(`*port#*`)`
- Low values indicate large distance
- High values indicate distance approaching ~4 inches
- Range is 4-30 inches. Result is approx $1/d^2$. Objects closer than 4 inches will appear to be far away.




# Optical Rangefinder



# Ultrasonic Rangefinder (Sonar)

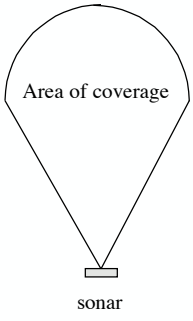


- Timed analog sensor
- Connect **red** to Expansion board
  - (upper deck) port #0
- Connect **gray** to Digital #7
- Access with function `sonar()`
- Returned value is distance in mm to closest object in field of view
- Range is approximately 30-2000mm
- No return (because objects are too close or too far) gives value of 32767
- Wait at least .03 seconds between `sonar()` calls




# Ultrasonic Sensors

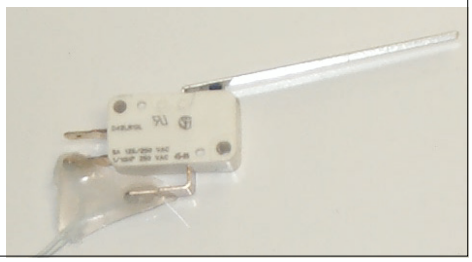
- Puts out a short burst of high frequency sound
- Listens for the echo
- Speed of sound is ~300mm/ms
- `sonar()` times the echo, divides by two and multiplies by speed of sound
- The sonar field of view is a 30º teardrop



# Touch Sensors

- Digital sensor
- Connect to ports 7-15
- Access with function `digital` (*port#*)
- 1 indicates switch is closed
- 0 indicates switch is open
- These make good bumpers and can be used for limit switches on an actuator








# Slot Sensors

- Digital sensor
- Connect to ports 7-15
- Access with function `digital` (*port#*)
- 1 indicates slot is empty
- 0 indicates slot is blocked
- These can be used much like touch sensors (if the object being touched fits in the slot)
- Special abilities when used as encoders

Botball

# Using Encoders

- Use the slot sensor (Handy Board only)
- Connect to ports 7,8,12,13 (*encoder#* is 0,1,2,3)

  `enable_encoder(`*encoder#*`);`
  - Only enable an encoder once...unless you disable it.

  `disable_encoder(`*encoder#*`);`

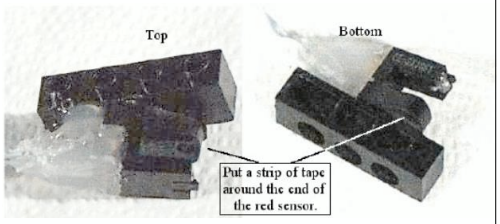
  `read_encoder(`*encoder#*`);`
  - returns the number of transitions

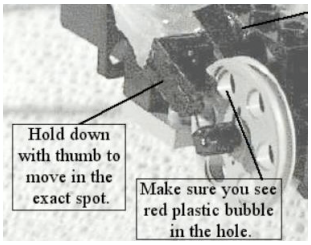
  `reset_encoder(`*encoder#*`);`
  - sets that encoder to 0
- Reflectance and slot sensors work best for encoders




# Mounting a slot sensor encoder

Carefully align sensor with encoder wheel



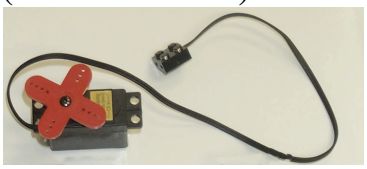
# Simple Encoder Program

```
void main(){
   int enc1, enc0;
   enable_encoder(0); /* turn on the encoders */
   enable_encoder(1);
   while(!stop_button())
   {
      enc0=read_encoder(0);  /* read each encoder */
      enc1=read_encoder(1);  /* and show values   */
      printf("Enc0=%d Enc1=%d\n",enc0, enc1);
      sleep(0.1);  /* wait a bit and do it again */
   }
}
```




# Handy Board DC Motors

- All motors are connected to HB using motor wire
- Micro-motor slow gear motor
- High-power gear motor (modified servo)



- High speed gear motors

# Handy Board Servo Motors

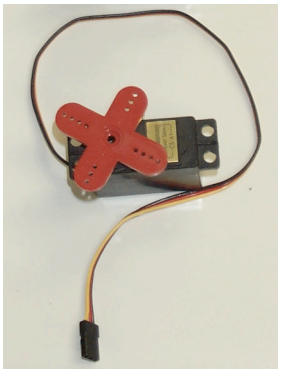
- When plugged in, black wire is on left, yellow on right
- Enable Servos:

```
init_expbd_servos(1);
```

- Disable Servos:

```
init_expbd_servos(0);
```

- To change servo positions, change values of: **`servo0, servo1,…,servo5`**
  *(these are examples of global variables)*
- Values should range between 100 and 3900
- Note: Servos acting weird or not working is the first sign of low battery






# Recharging Kit Batteries

- Handy Boards should be charged through the interface board
  - Interface board charging can be done in normal or Zap mode.
    - Normal charge lights yellow light on interface board when working, and does a trickle charge -- you can leave the board charging in this mode indefinitely. A full charge takes about 12 hours, though you can use the board earlier.
    - **ZAP fully charges in 2 hours, and will damage batteries after that. ONLY recommended while working with the robot with a dead battery or in tournament emergencies,**

# Experimenting With Motors and Sensors Using the IC Interaction Window

- Motors

```
fd(3);
bk(3);
off(3);
motor(3,75); motor(2, -50);
```

- Sensors

```
digital(7);
analog(6);
sonar();
```





# Testing Your HB, Sensors & Motors

- Download **hbtest.ic** (from the IC folder)
- connect and test motors
- connect and test servos (one at a time)
- connect and test digital sensors
- connect and test analog sensors (be sure you have the right type of sensors in the right ports)
- connect and test sonar sensor

# Lego RCX

- Simple controller
  - Similar memory and processor to Handy Board
    - Three sensor inputs
    - Three motor ports
  - Communicates to your computer via the IR tower
  - RCX powered by 6 alkaline AA batteries
  - IR tower powered by 9v alkaline battery
  - Gray DB9 to DB9 cable connects tower to serial port
  - Mac Serial port users, use HB cable, the 25-9 pin connector and then the gray DB9-DB9 cable



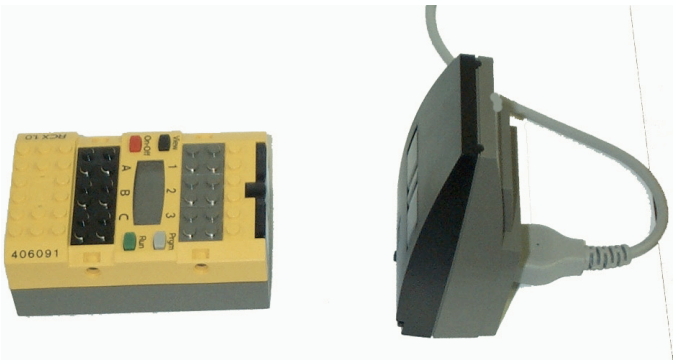


# The RCX Controller

- Your kit may also include a LEGO RCX controller
- Use the “switch controller” menu item from the IC settings menu to bring up the picture of the RCX and select it
- If your RCX does not say IC 4 when you turn it on, then you will need to download the firmware

# RCX Setup

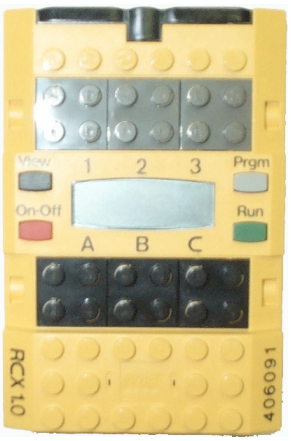
To your serial port





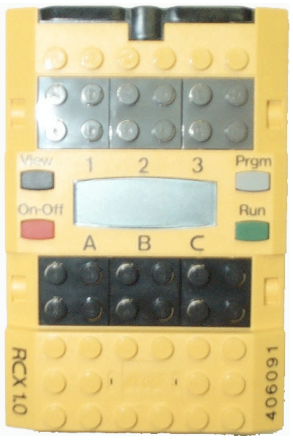


# RCX Ports




- Ports 1, 2, & 3 are the sensor ports
- Ports A, B & C are the motor ports (referred to in code as A, B & C (1, 2, & 3 also work) e.g., `fd(A);` or `fd(1);` turns on the motor on port A)

# RCX Buttons




- Power button turns RCX On and Off
- Run Button runs the main function ("crash appears on screen if no main is loaded")
- View button displays sensors port values if program is NOT running
- `view_button()` and `prgm_button()` --access from your program (like start & stop)




# IR Tower Battery

Exposed connector means Bad


Good

- Be sure to insert the battery all the way
- Connectors fit into the left wall
- If your communication does not work, check battery

# RCX Cool Functions in IC

- `battery_volts()` returns the level of the RCX batteries
- `brake(`*n*`)` stops motor *n* (e.g., 1, 2 or 3) quickly (more quickly than `off(`*n*`)` )
- `allbrake()` same as above
- `light(`*p*`)` returns value of light sensor, with emitter on, connected to port *p*
- `light_passive(`*p*`)` returns value of light sensor connected to port *p*
- `poweroff()` turns off the RCX





# RCX functions in IC

- RCX does not support…
  - `analog(`*p*`)`
  - math functions other than + - / *
  - sonar
  - servo functions, encoder functions
- RCX supports very limited `printf` capabilities (%d works; no %f; no \n ever needed in RCX; characters do not look very good)

# RCX Sensors

- The RCX can only use LEGO Mindstorms sensors
  - Touch sensor use `digital(port#)`
  - Light sensor use `light(port#)` to get areflectance reading or `light_passive(port#)` to get a passive light reading

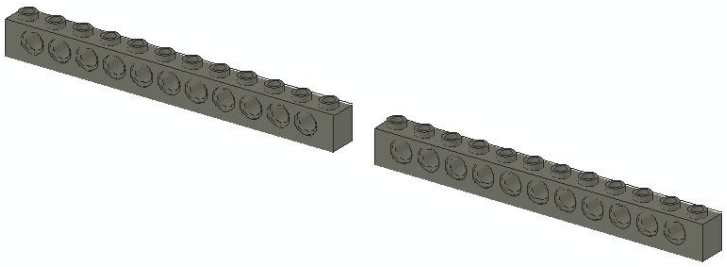




# A Simple LEGO Robot

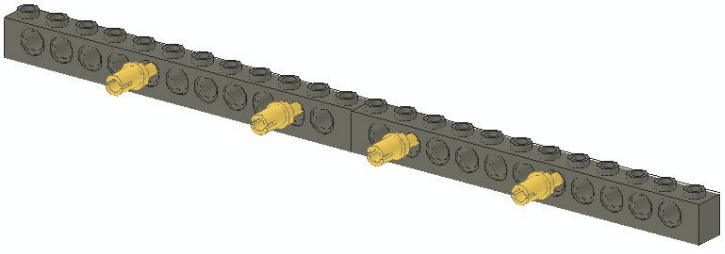
- The next several slides give step by step assembly instructions for a simple robot chassis
- This robot may be used with either the RCX or Handy Board controller (a few extra blocks will be needed to secure the HB to the chassis)
- The colors of your parts may vary

71





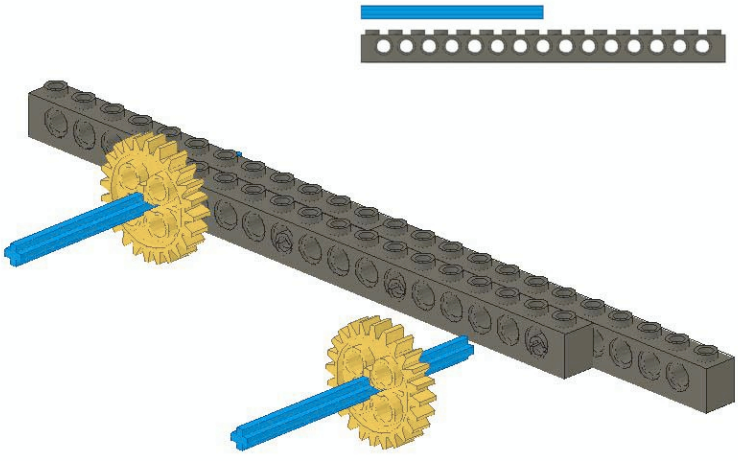
72



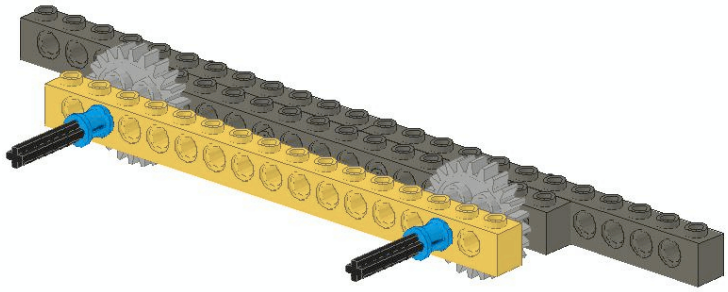
73



74

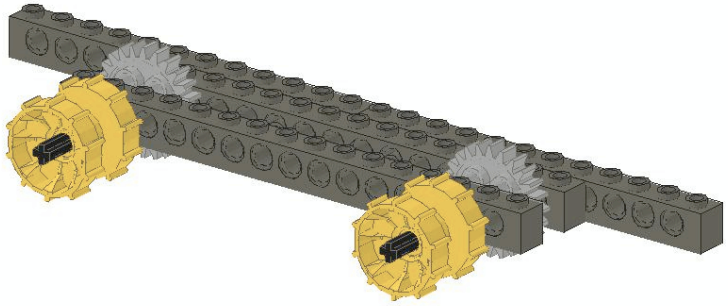


75

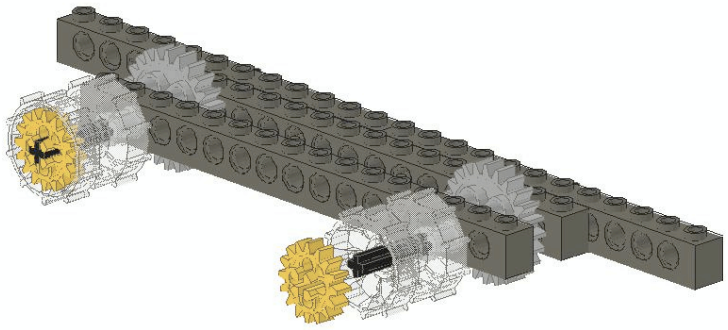




76

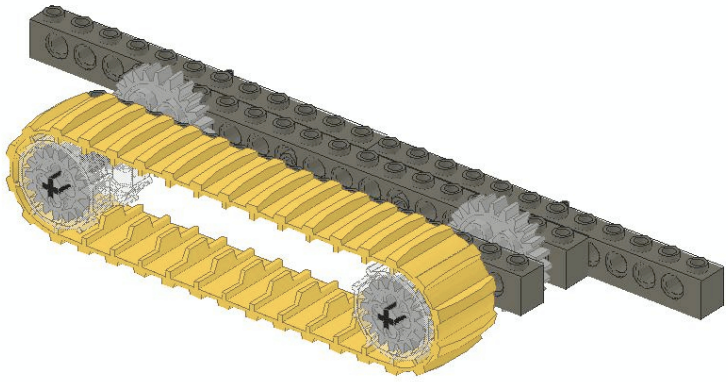


77

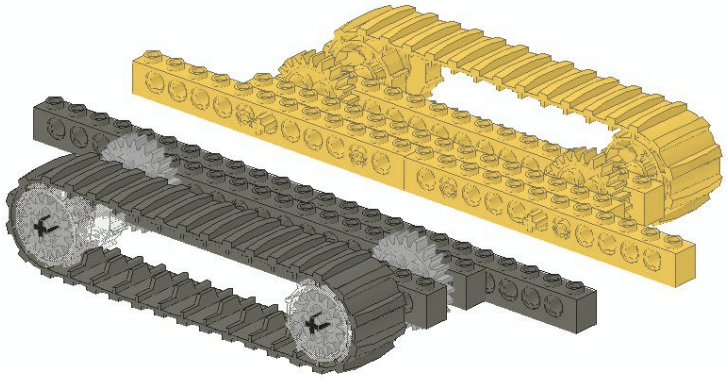




78



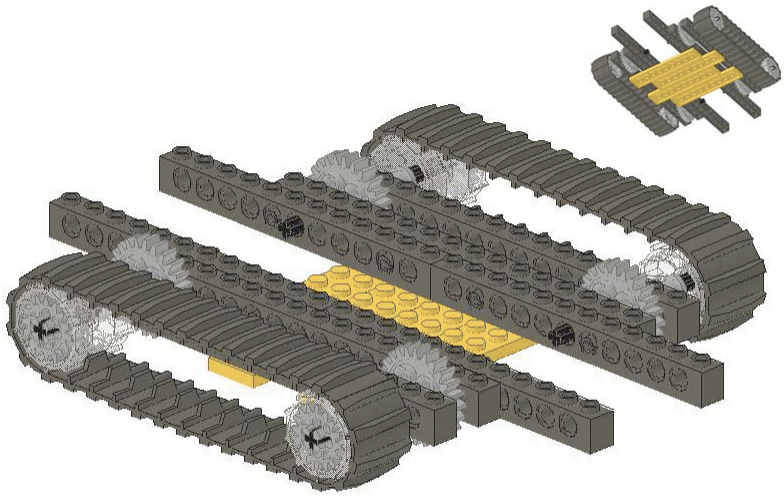
79



Botball

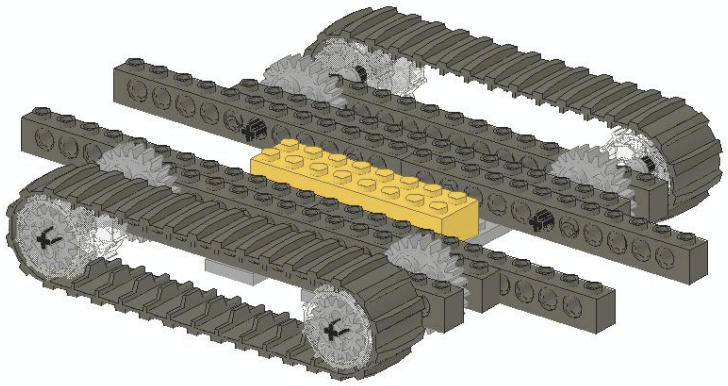


80



Botball

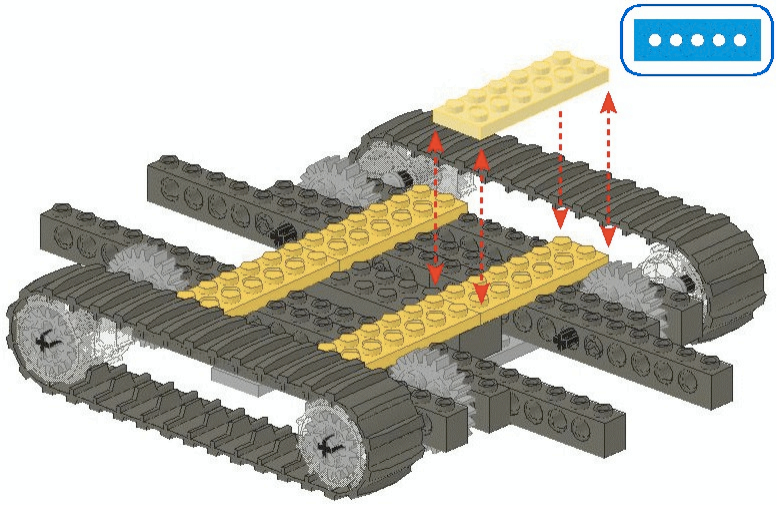
81



Botball

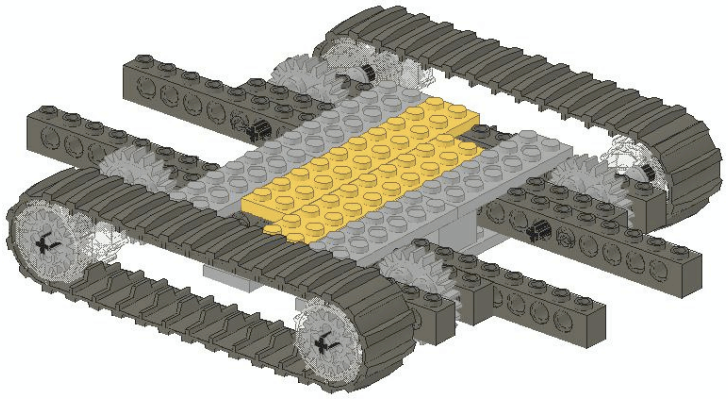


82



Botball

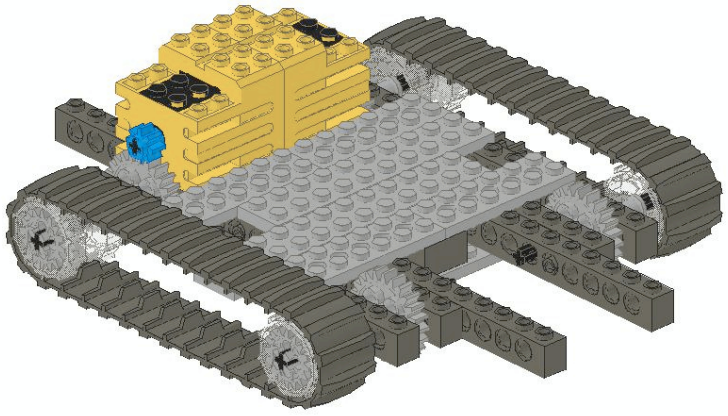
83



Botball

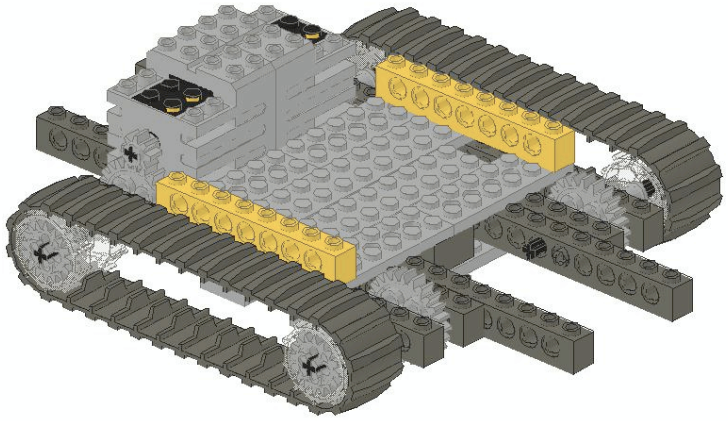


84



Botball

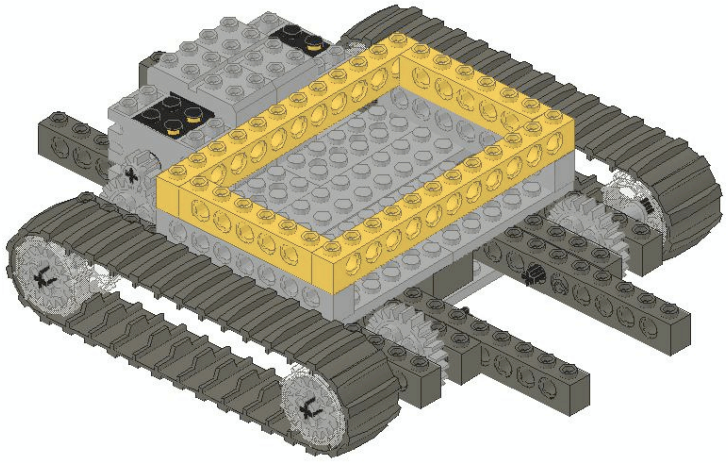
85



Botball

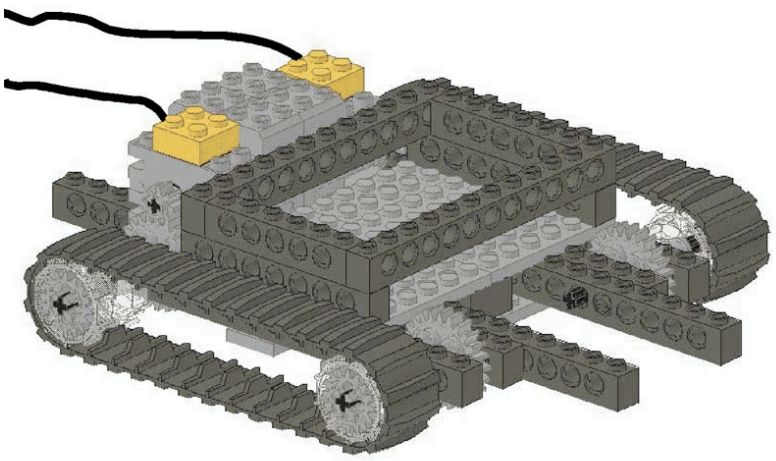


86



Botball

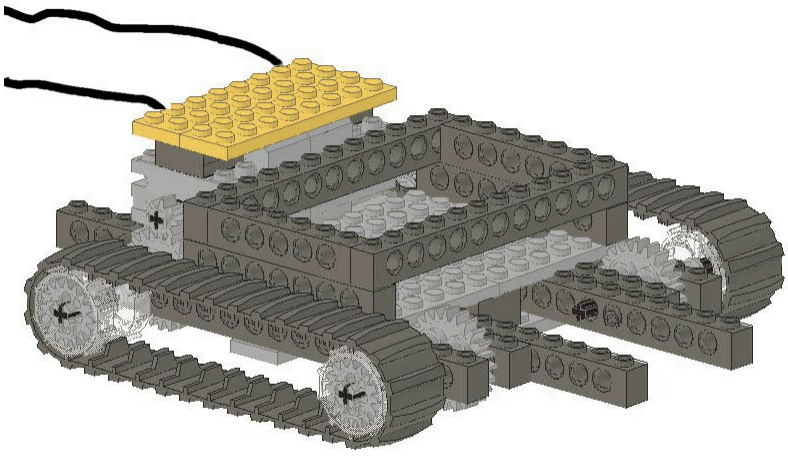
87



Botball



88



Botball

# A Simple Bumper