

The Open University of Hong Kong

School of Science and Technology

Computing Programmes

Parallax ActivityBot

Robotics Programming Short Course Workbook

Developers and Editors: Ms. Yvonne Lam, Ms. Wing Go, and Dr. Andrew Lui

Version: 2014

Content, photos, and figures adapted from learn.parallax.com under licence

Chapter 0

Objective

- ✓ To learn simple operations of Propeller Activity Board and SimpleIDE

1. Propeller ActivityBoard

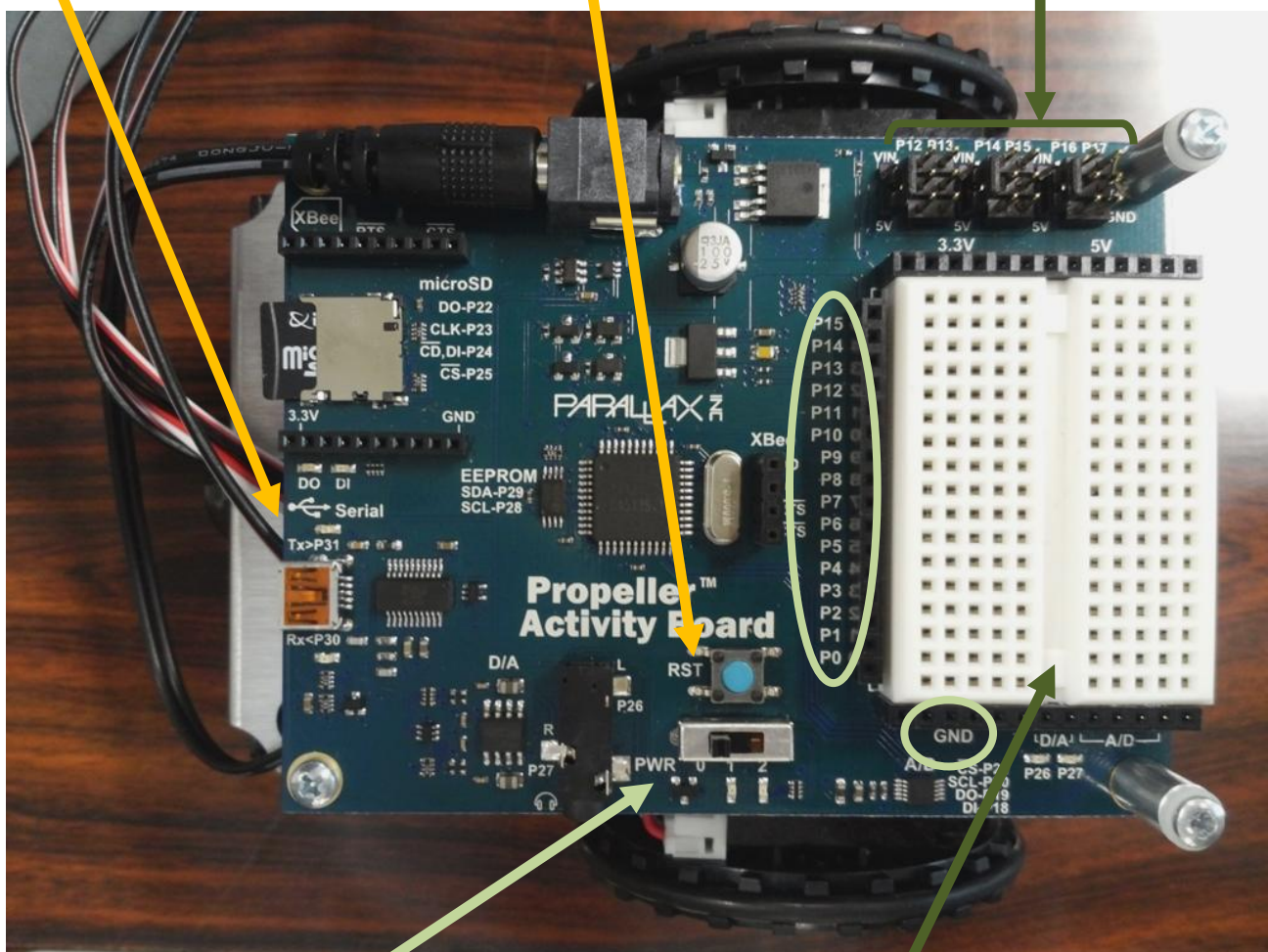
You will need to insert some devices and USB cable to this ActivityBoard in order to run a program. Here is a general introduction:

Serial:
connect robot and computer
using USB cable

RST (reset button):
Run /stop program

Power supply:

- Supply power from batteries
- Right servo to Pin13, left servo to P12
- Right wheel to Pin15, left wheel to Pin14



PWR (Power switch):

- Position 0 - on /off
- Position 1 - load program
- Position 2 - run program

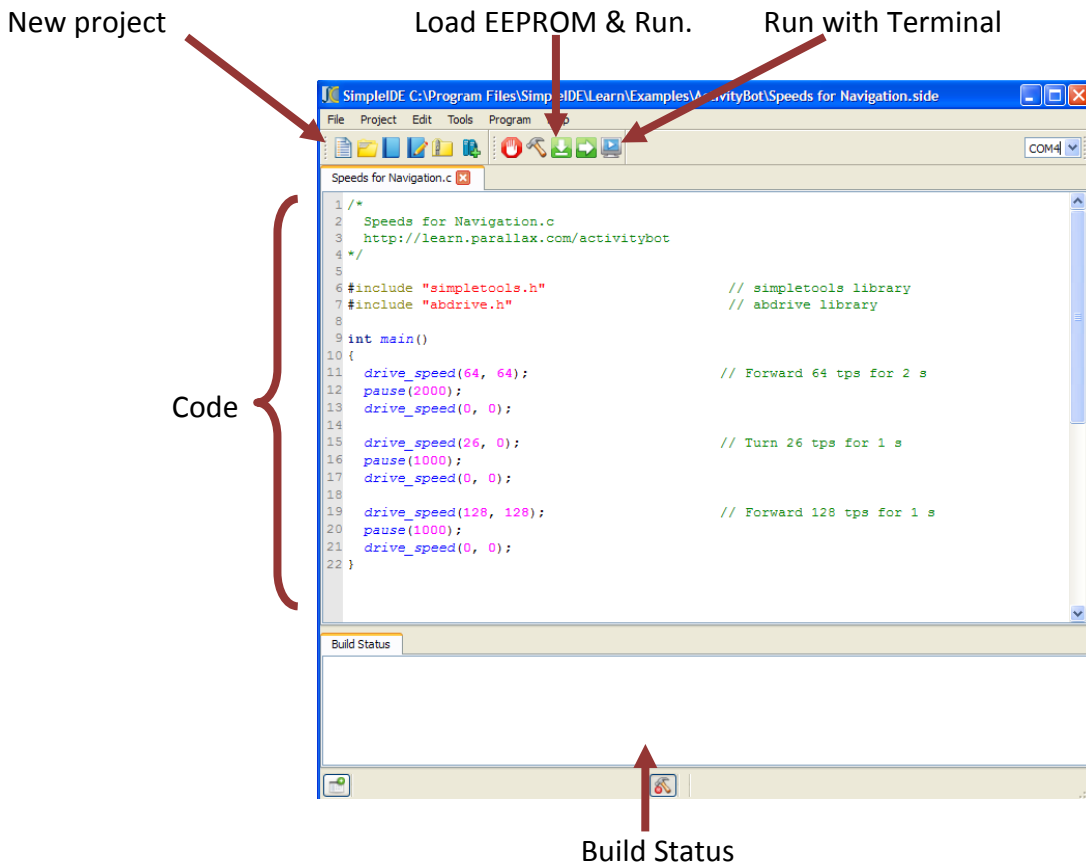
Breadboard:

- P0 – P15 – Pin0 -15, supply power from Serial
- GND(ground)

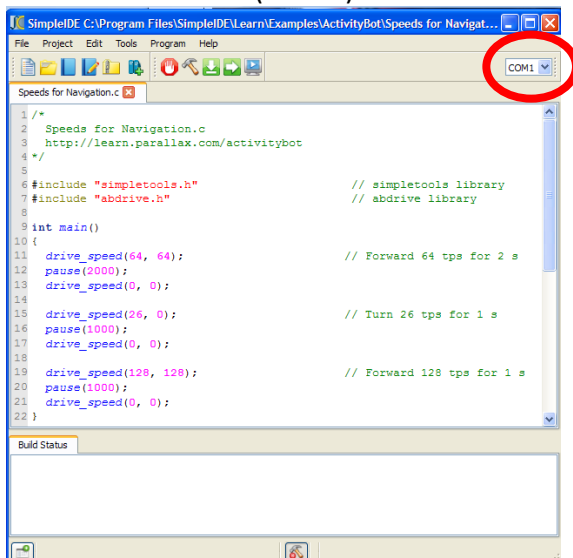
2. SimpleIDE

We usually use SourceEdit as our compiler in C programming lessons but this time, we are going to use SimpleIDE. SimpleIDE is specially designed for beginners to learn constructing programs to run the Propeller ActivityBoard. These are the very few buttons and display you need to know.

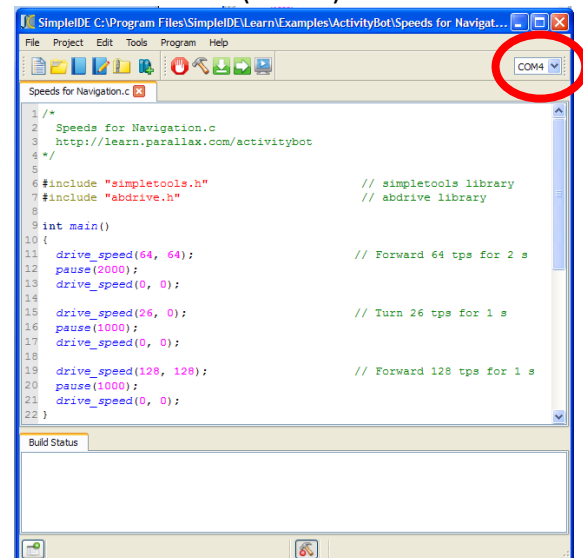
General Display:



USB Serial Port number:
Before connection (COM1)

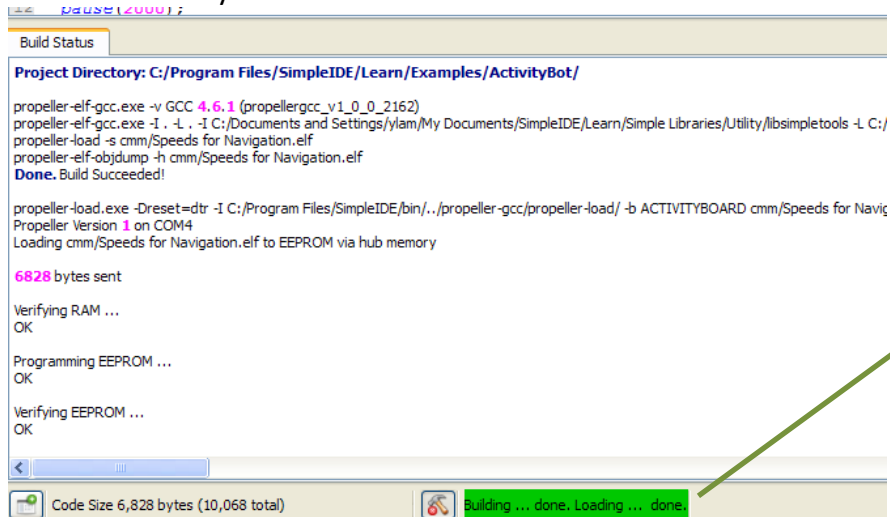


After connection (COM4)



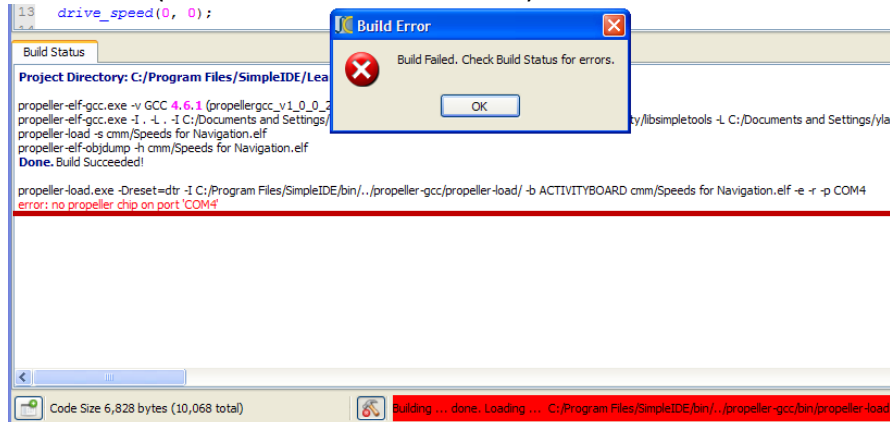
Build Status:

Build successfully



Building and loading
successfully

Build Failed(USB Serial Connection Error)



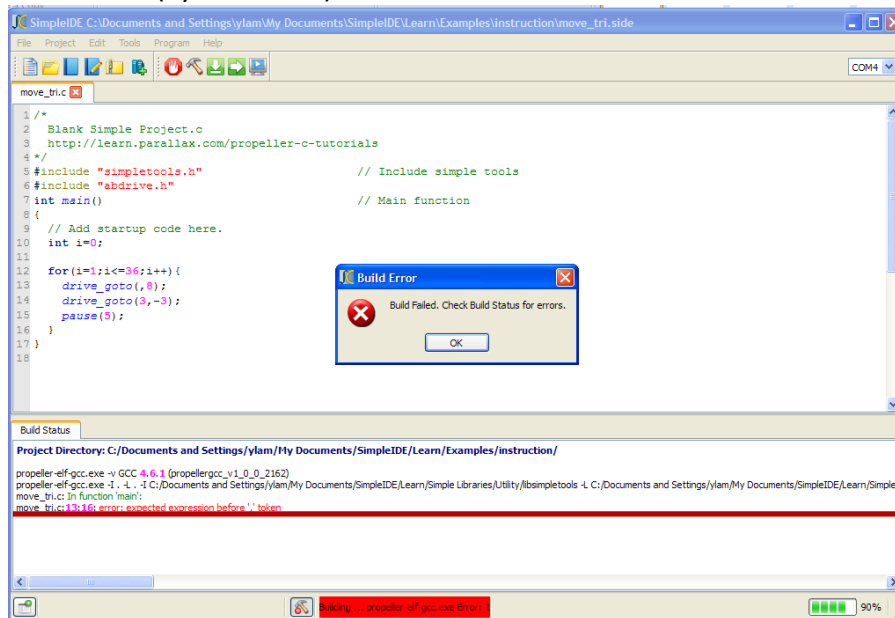
Error description:

error: no propeller chip on port 'COM4'

How to fix:

- Check if you have connect cable to your robot
- Try to change another port

Build Failed(Syntax Error)



Error display format

File name: line number: column number: error description

Move_tri.c:13:16:error:expected expression before ',' token.

How to fix:

- Check you syntax in program

Chapter 1

Objective

- ✓ To construct, test and calibrate a Parallax ActivityBot
- ✓ To run simple function- moving forward/ backward, turning in different degree

1. Test the Encoder Connections

In this activity, right and left wheels are checked if connected correctly.

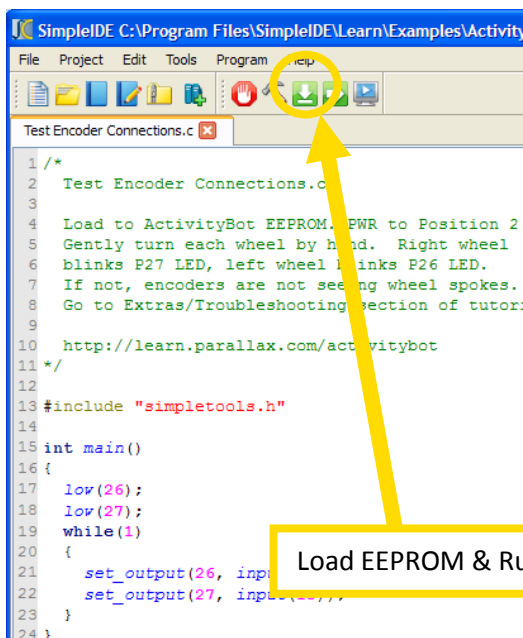
Steps:

1. Set the robot's PWR switch to Position 2.
2. Click SimpleIDE's Open Project button.
3. Browse Documents/SimpleIDE/Learn/Examples/ActivityBot/Test Encoder Connections.
4. Click Load EEPROM & Run. If you see the message "Build Failed. Check Build Status for errors.", please check the status on upper right corner, try to switch to another status(e.g. COM4).
5. Rotate the ActivityBot's right wheel gently. This should make the P27 LED turn on and off as you rotate the wheel.
6. Repeat for the left wheel. Turning the left wheel should make the P26 LED turn on and off.

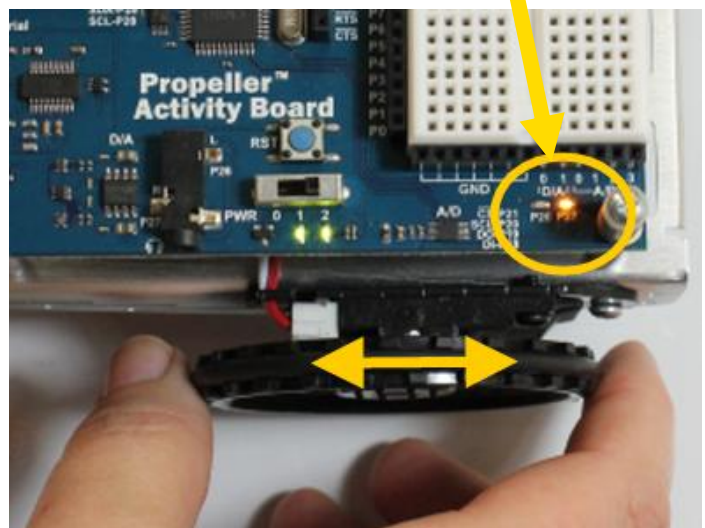


PWR switch

P26, P27 LED



Load EEPROM & Run.



If the ActivityBot doesn't work, check if...

P26 Light stays off while turning the right wheel.

1. The right encoder cable may be plugged into the P14 servo pin backwards.
2. 20 k resistor (red-black-brown) may not be making contact at either the P14 or 3.3 V socket.
3. The right encoder may be mounted with the sensor facing the chassis instead of facing the wheel spokes.

P27 Light stays off while turning the left wheel.

1. The left encoder cable may be plugged into the P15 servo pin backwards.
2. 20 k resistor (red-black-brown) may not be making contact at either the P15 or 3.3 V socket.
3. The left encoder may be mounted with the sensor facing the chassis instead of facing the wheel spokes.

P27 light instead of P26 light blinks while wheel turning the right wheel (or vice versa).

1. The encoder cables are swapped. Switch the encoder cables plugged into P14 and P15.

P26 or P27 light stays on while turning wheel.

1. Resistor connecting P14 or P15 socket to 3.3 V socket is too small. It should be 20 k-ohm (red-black-orange-gold). This resistor came in the bag with the encoder parts, not with the rest of the resistors in the kit.

2. Calibration

Before running any other example programs, your ActivityBot needs to be calibrated. This is a one-time calibration that the *abdrive library* needs for measuring and correcting distances and speeds, using information from the ActivityBot encoders.

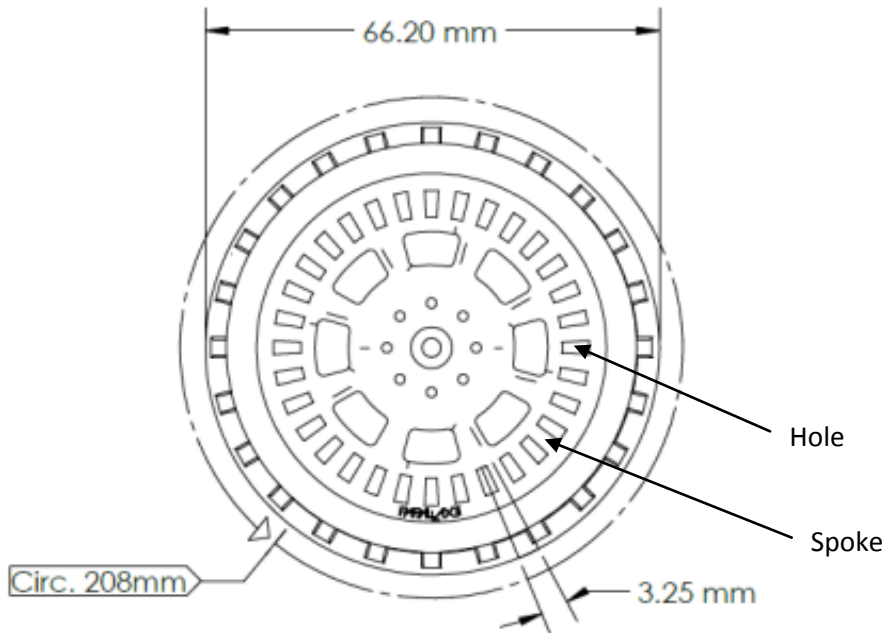
Steps:

1. Set the ActivityBot's PWR switch to position 1.
2. Click SimpleIDE's Open Project button.
3. Open Documents/SimpleIDE/Learn/Examples/ActivityBot/ActivityBot Calibrate.
4. Click the Load EEPROM & Run button. If you see the message "Build Failed. Check Build Status for errors.", please check the status on upper right corner, try to switch to another status.
5. When the program is finished loading, the P26 and P27 lights will turn on. When they come on, turn off the robot's power (Slide PWR switch to position 0).
6. Disconnect the ActivityBot from its programming cable and set it in a 1 meter, obstacle-free, smooth floor.
7. Set the power switch to 2 and move back to give it room to spin in place and slowly roam while it gathers wheel speed data.
8. Leave it alone until the P26 and P27 lights turn off (about 2 minutes). After that, calibration is complete and you can turn the power off.

3. Move forward and backward

"Tick" indicates a transition from either spoke detected to hole detected, or vice-versa. The Propeller ActivityBot wheel has 32 spokes, separated by 32 spaces, for a total of 64 ticks. If the wheel turns 1/64th of a turn, it will travel 3.25 mm.

In other words, a wheel consists of 64 ticks and 1 ticks = 0.325cm



Normally, we will set the parameter in `drive_speed()` to positive in order to run forward and negative to run backward.

Libraries included:

1. `abdrive.h`
2. `simpletools.h`

Steps:

1. Set the ActivityBot's PWR switch to position 1.
2. Click SimpleIDE's Open Project button.
3. Open Documents/SimpleIDE/Learn/Examples/ActivityBot/Test for Trim.
4. Click the Load EEPROM & Run button. If you see the message "Build Failed. Check Build Status for errors.", please check the status on upper right corner, try to switch to another status.
5. After the program finishes loading, set PWR switch to position 0 and disconnect the USB cable.
6. While holding the reset button down, set the PWR switch to position 2, and set the robot on a hard, smooth floor.
7. Release the reset button, and the robot should travel forward at a speed of 32 ticks per second for 8 seconds and finally stop.

*To move backward, set the parameter in function `drive_speed ()` to negative.

Functions to be customized:

1. `drive_speed(right wheel tick per second, left wheel tick per second)`
Set the activityBot to move in a certain speed, measured in tick per second. Higher the number of ticks, faster the robot runs.
2. `pause(milliseconds)`
Retain the former action for assigned millisecond.

Function for reference:

1. `drive_trimSet(0,0,0)`

Clear any settings a robot might already have.

Referring to Test for Trim.side

```
#include "simpletools.h"
#include "abdrive.h"

int main() {
    drive_trimSet(0, 0, 0);
    drive_speed(32, 32);
    pause(8000);
    drive_speed(0, 0);
}
```

TIPS

Except `drive_trimSet(0, 0, 0)`, the two functions `drive_speed()` and `pause()` are ready for you to modify.

Change the parameters in `drive_speed()` to 96, you'll notice the robot run faster than before. *Never set the parameters to higher than 128, it may damage the robot servo.

Change the parameter in `pause()` to 2000, you'll notice the robot run for 2 seconds instead of 8 seconds.

Try the modified "Test for Trim.side" below:

```
#include "simpletools.h"
#include "abdrive.h"

int main() {
    drive_trimSet(0, 0, 0);
    drive_speed(96, 96);
    pause(2000);
    drive_speed(0, 0);
}
```

4. Move for certain distance

Velocity is speed, but it can be either positive (forward) or negative (backward). Speed itself is just velocity without the sign. So, if you are going backwards at a speed of 5 km/hr, you could also call that a velocity of -5 km/hr.

Here is an equation you can use to calculate distance (s) given velocity (v) and time (t).

$$s = v \times t$$

Example: If a wheel goes 64 ticks per second for 2 seconds, then $v = 64$ ticks per second and $t = 2$ seconds. So:

$$s = 64 \text{ ticks/second} \times 2 \text{ seconds} = 128 \text{ ticks.}$$

For a distance in centimeters, remember that a tick is 0.325 cm, so the wheel travels:

$$s = 128 \text{ ticks} \times 0.325 \text{ cm/tick} = 41.6 \text{ cm.}$$

Libraries included:

1. `abdrive.h`
2. `simpletools.h`

Steps:

1. Set the ActivityBot's PWR switch to position 1.
2. Click SimpleIDE's Open Project button.
3. Open Documents/SimpleIDE/Learn/Examples/ActivityBot/Speeds for Navigation.
4. Click the Load EEPROM & Run button. If you see the message "Build Failed. Check Build Status for errors.", please check the status on upper right corner, try to switch to another status.
5. After the program is done loading, set PWR switch to position 0 and disconnect the USB cable.
6. Set the power switch to 2 on a hard, smooth floor. The robot should move for 64 ticks per second for 2 seconds, turn right and travel at 128 ticks per second for 1 second.

Functions to be customized:

1. `drive_speed(right wheel tick per second, left wheel tick per second)`
Set the activityBot to move in a certain speed, measured in tick per second. Higher the number of ticks, faster the robot runs.
2. `pause(millisecond)`
Retain the former action for assigned millisecond.

Referring to Speeds for Navigation .side

```
#include "simpletools.h"
#include "abdrive.h"

int main(){
    drive_speed(64, 64);
    pause(2000);
    drive_speed(0, 0);

    drive_speed(26, 0);
    pause(1000);
    drive_speed(0, 0);

    drive_speed(128, 128);
    pause(1000);
    drive_speed(0, 0);
}
```

TIPS

Did you notice one of the servos got zero speed? This can make the robot turn right slightly.

Try to modify `drive_speed(26, 0)` to `drive_speed(0, 26)` in "Speeds for Navigation.side", you'll notice your robot turn left instead of turn right:

```
#include "simpletools.h"
#include "abdrive.h"

int main() {
    drive_speed(64, 64);
    pause(2000);
    drive_speed(0, 0);

    drive_speed(0, 26);
    pause(1000);
    drive_speed(0, 0);

    drive_speed(128, 128);
    pause(1000);
    drive_speed(0, 0);
}
```

5. Turning different degrees

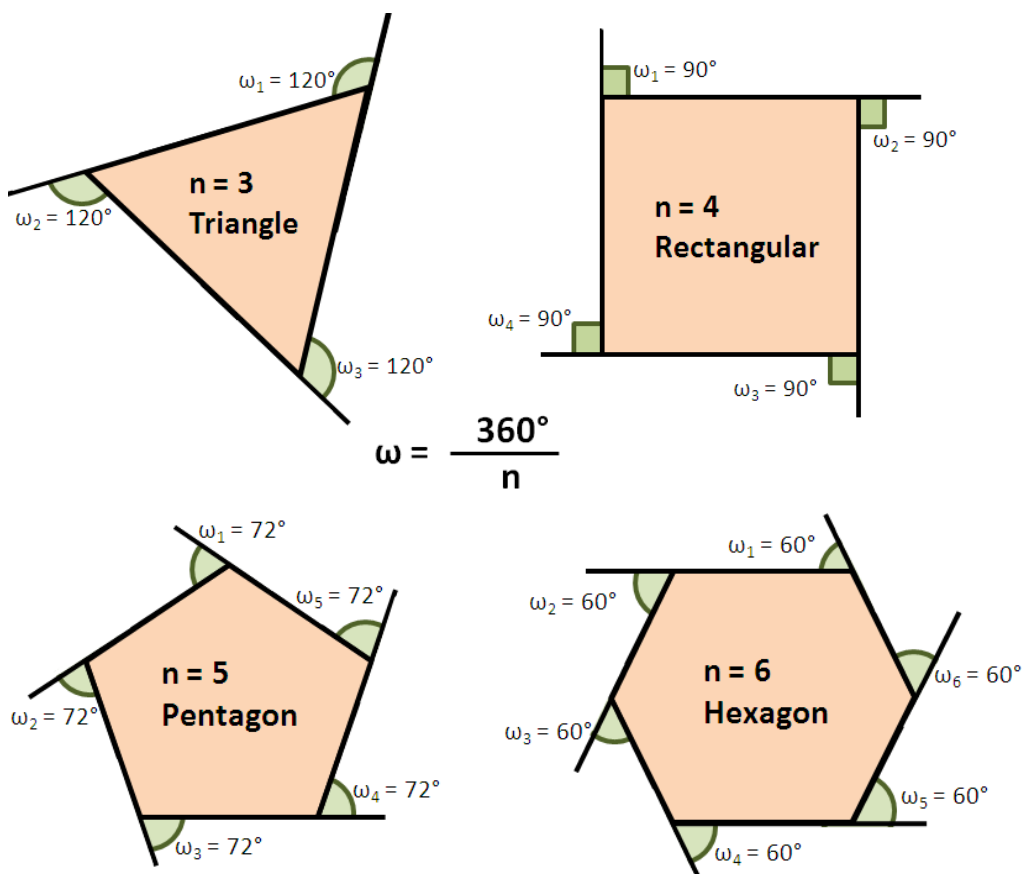
A polygon is a closed shape that has n -number of vertices and n -number of sides ($n \geq 3$). Specifically, something that is called a regular polygon is one in which all sides are of equal length.

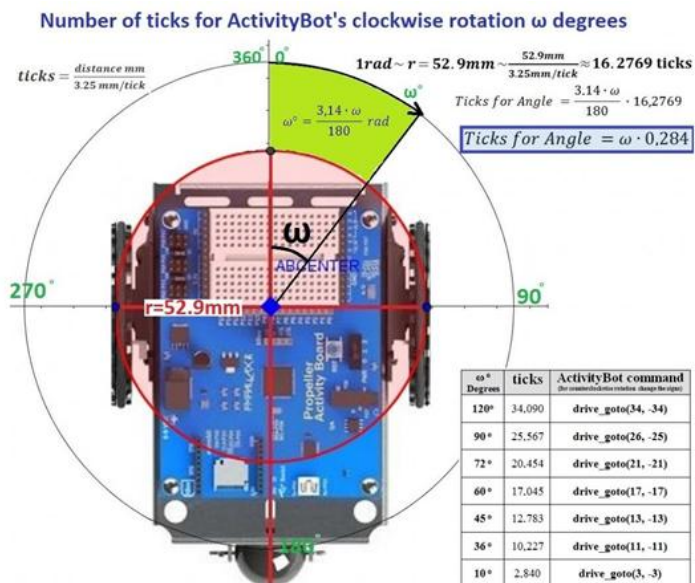
Some examples of regular polygon types are below:

- the equilateral triangle ($n = 3$)
- the square ($n = 4$)
- the regular pentagon ($n = 5$)
- the regular hexagon ($n = 6$), and etc.

Regular polygons with ($n \geq 36$) are of special interest, because in terms of shape they are very close to the circle. Although they are not real circles (real circles lack any vertices), in practice these polygons have so many vertices that are close enough together that they can represent a circle with high precision.

An external angle (or exterior angle) is the angle that is formed outside of the polygon between one side and a line extended from another side adjacent to it. It is essential to know the external angle of the regular polygon in order to give the robot the appropriate command to turn properly.





When the ActivityBot rotate, it moves its wheels based on the inner circle (radius: 52.9 mm).

$$r = 52.9 \text{ mm} = 52.91 / 3.25 \text{ ticks} = 16.2769 \text{ ticks}$$

Changing rad to degrees, we take the formula:

$$\text{Ticks for Angle} = \omega \cdot 0.284$$

So if we want to program the ActivityBot to rotate ω degrees we can multiply by 0.284 in order to find the appropriate number of wheel ticks.

For example, we can have the robot rotate for 90 degrees:

$$90^\circ \cdot 0.284 \text{ ticks} = 25.56 \text{ ticks} = 26 \text{ ticks (rounded up)}.$$

Here is a degree table summarizing the above. You can simply make use of the predefined parameters in the table.

| ω° Degree | Ticks | ActivityBot Command |
|--------------------------|-------------|---------------------|
| 120 | 34.090(~34) | drive_goto(34,-34) |
| 90 | 25.567(~26) | drive_goto(26,-25) |
| 72 | 20.454(~21) | drive_goto(21,-21) |
| 60 | 17.045(~17) | drive_goto(17,-17) |
| 45 | 12.783(~13) | drive_goto(13,-13) |
| 36 | 10.227(~11) | drive_goto(11,-11) |
| 10 | 2.840(~3) | drive_goto(3,-3) |

Steps:

1. Set the ActivityBot's PWR switch to position 1.

2. Enter the code below and make sure load it to your robot with the "Load EEPROM & Run" button.

For loop:

```
#include "simpletools.h"
#include "abdrive.h" }
```

Libraries

Decide how many times the robot change direction. For a triangle, 3 lines are needed to form the shape. Therefore, the segment should repeat 3 times.

*If you don't understand the concept of a "for loop", please refer to section3b in Chapter2.

```
int main() {
```

```
int n;
```

```
for(n = 1 ; n <= 3 ; n++) {
```

```
drive_goto(64,64);
```

```
pause(200);
```

```
drive_goto(34,-34);
```

```
pause(200);
```

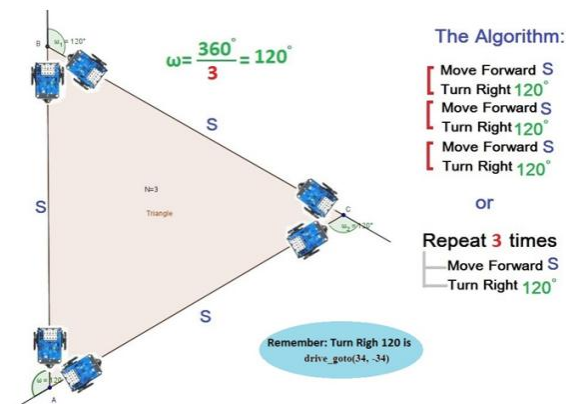
```
}
```

```
}
```

Travel straight at the speed of 64ticks per second for 0.2 second

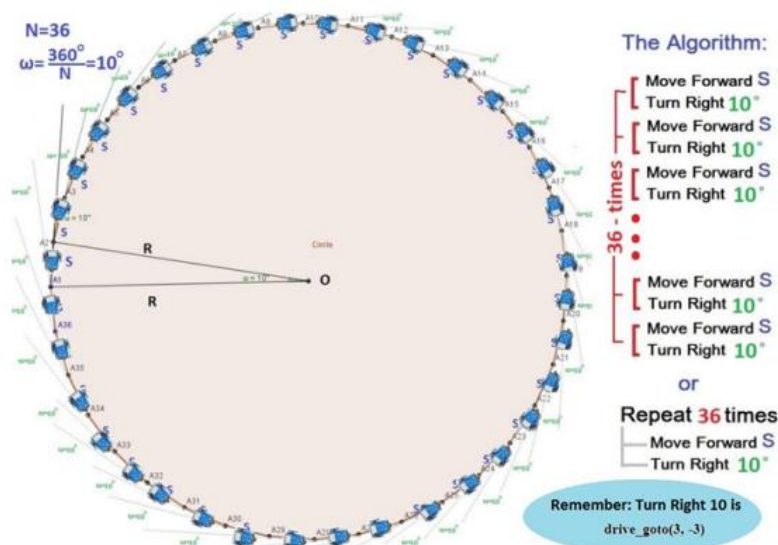
Change direction, turn right for 120°

3. After the program is done loading, set PWR switch to position 0 and disconnect the USB cable.
4. Set the ActivityBot on the floor and switch the PWR to position 2. The robot should travel forward and change direction for 3 times. The route should look like a triangle.



Now, try another program to make the route become a circle. Change the code into the following:

| | | |
|--|---|--|
| | <code>#include "simpletools.h"</code> | |
| <i>For loop:</i> | <code>#include "abdrive.h"</code> | |
| <i>Decide how many times the robot change direction. For a circle, there should not be any vertices but polygons with more than 36 vertices will be very close to a circle. Therefore, the segment should repeat 36 times.</i> | <pre> int main(){ int n; for(n = 1 ; n <= 36 ; n++){ drive_goto(64,64); pause(200); drive_goto(3,-3); pause(200); } </pre> | <p><i>Travel straight at the speed of 64ticks per second for 0.2 second</i></p> <p><i>Change direction, turn right for 10°</i></p> |



If you want the robot move in different shapes, try to make use of the degree table above.

Functions to be customized:

1. `drive_goto(right wheel tick per second, left wheel tick per second)`
2. `pause(millisecond)`
Retain the former action for assigned millisecond.

Chapter 2

Objective

- ✓ To learn simple programming techniques- procedures, condition, iteration
- ✓ To test some simple functions using various devices (light sensor, ultrasonic sensor)
 1. Procedure(step by step)
 - a. Blink Light(procedure)
 2. Condition
 - a. If
 - i. moveForward
 - b. If...else
 - i. lightsense
 3. Iteration
 - a. While loop
 - i. Blink Light(while_loop)
 - b. For loop
 - i. buzz(procedure_for_loop)

1. Procedure (step by step)

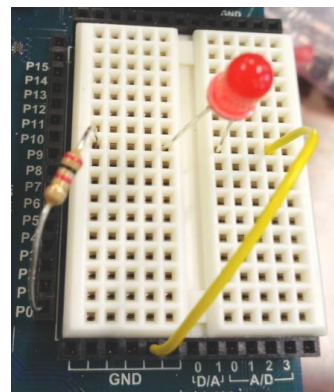
We usually do things step by step. For example, if we want to eat an omelet, we'll first take an egg and a piece of ham from fridge, then whisk them and finally put them in oven. You can see cooking an omelet consists of 3 steps. In programming, we also do the same thing. If we want a light to light for two second and then go off, we need to provide electricity to the light and turn off power after 2 seconds. The following program will illustrate how the LED light bulb and functions work.

Steps:

1. Connected resistor from Pin0.
2. Plug a LED (light-emitted diode) (longer side positive).
3. Run program.

Functions to be customized:

1. `high(pin number)`
Supply power to Pin.
2. `pause(millisecond)`
Retain the former action for assigned millisecond.
3. `low(pin number)`
Stop supply power to Pin.



Referring Blink Light(procedure).side


```
#include "simpletools.h"

int main() {
    high(0);
    pause(2000);
    low(0);
    pause(2000);
}
```

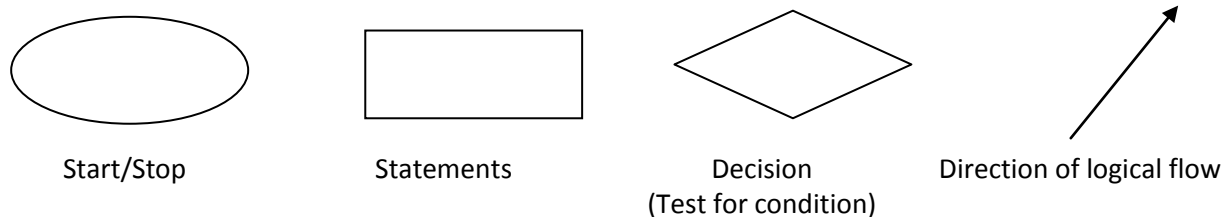
Supply power to Pin0

Retain the former action for 0.2 second

Stop supply power to Pin0

2a. Condition If

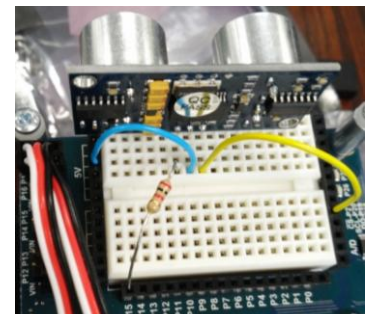
Before learning how to make a decision, we need to know some simple symbols representing start, stop, statements, and conditions of a program in a flow chart.



We make lots of decisions every day and each decision is made under certain condition. In programming, we use the keyword “if” to represent the condition and the statement after “if” is the decision. Referring to the program “moveForward.side”, we will come across the condition “if a sensor detects an obstacle 5 cm before it” and the decision “the robot stop”. In this activity, the robot keeps moving forward until an obstacle is detected 5cm in front of the sensor.

Steps:

1. Connect right servo to Pin13, left to Pin12.
2. Connect a 2 k-ohm resistor (red-black-red) from Pin15 to sensor”SIG”.
3. Supply power from 5V to sensor”5V”.
4. Connect ground (GND) to sensor”GND”.
5. Install sensor.
6. Run program.



Functions to be customized:

1. `servo_angle(pin number, degree)`
Degree is set from 0 to 3600 to have a fine adjustment.
Set the angle of servo (wheels).
2. `ping_cm(pin number)`
Obtain distance between obstacle and sensor.
3. `servo_stop()`
Stop servo process (stop moving).

Variables:

4. `cmDist`
Store the distance between obstacle and sensor.

```
/* moveForward.c */
```

```
#include "simpletools.h"
```

```
#include "servo.h"      } Include servo header  
#include "ping.h"      } Include ping header
```

```
int main() {
```

```
    while(1){
```

```
        servo_angle(12, 1800); → P12 left servo to 180 degrees
```

```
        servo_angle(13, 0); → P13 right servo to 0 degrees
```

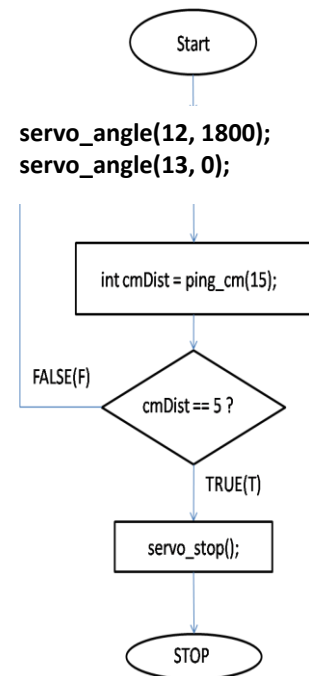
```
        int cmDist = ping_cm(15); → get distance for sensor
```

```
        print("cmDist = %d\n", cmDist); → print distance
```

```
        if(cmDist == 5){  
            servo_stop();  
        }
```

```
    }  
}
```

} Stop robot from moving if distance between obstacle and robot is less than 5cm



2b. if...else

Usually, we can have two outcomes when a condition is met and not. For example, if Mary is seriously sick, she will consult a doctor, if not, she will just rest at home. The outcome “consult a doctor” is followed when the condition “Mary is seriously sick” is met. The outcome “just rest at home” is followed when the condition is not met (Mary is not seriously sick). In programming, we use the keyword “else” to present a new outcome if the condition is not met. The following program will illustrate the condition “If a light sensor is masked (no light is given)” and the outcomes “the light bulb (red) turns on” and “ else the bulb goes off”.

Steps:

1. Connect a 220 ohm resistor (red-red-brown) from Pin0.
2. Connect a 0.01 μ F capacitor (red) to resistor.
3. Connect a phototransistor (transparent, shorter head) to capacitor.
4. Connect ground (GND) to phototransistor.
5. Connect a resistor from Pin15 to a bulb.
6. Connect the bulb to ground (GND).

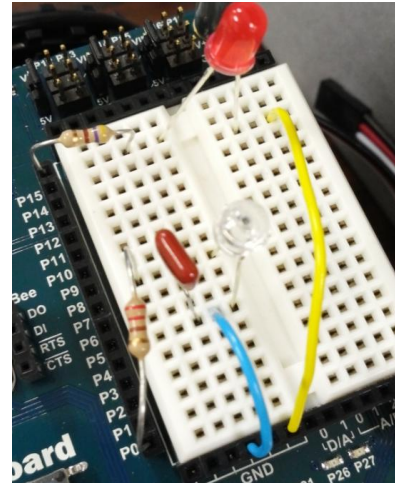
7. Run program.

Functions to be customized:

1. `high(pin number)`
Supply power to Pin.
2. `pause(millisecond)`
Retain the former action for assigned millisecond.
3. `low()`
Stop supply power to Pin.
4. `rc_time (pin number,1)`
Measure time for voltage to decay to 1.65V.
1= voltage on I/O pin larger than 1.65V.

Variables:

1. `t`
Store the time for voltage to decay to 1.65V.



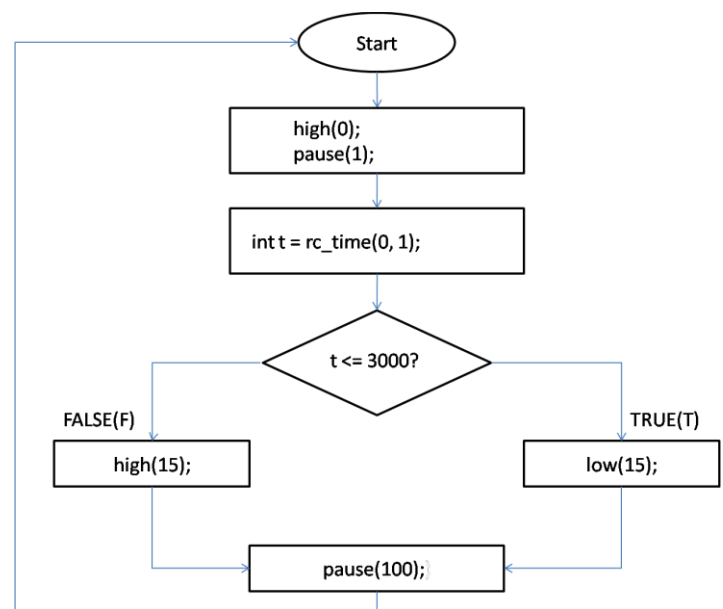
```
/*lightsense.c*/
```

```
#include "simpletools.h"
```

```
int main() {
```

```
    while(1) {  
        high(0);  
        pause(1);  
        int t = rc_time(0, 1);  
        printf("t = %d\n", t);  
        if(t<=3000)  
            low(15);  
        else  
            high(15);  
        pause(100);  
    }  
}
```

Measure time



3. Iteration

3a. While loop

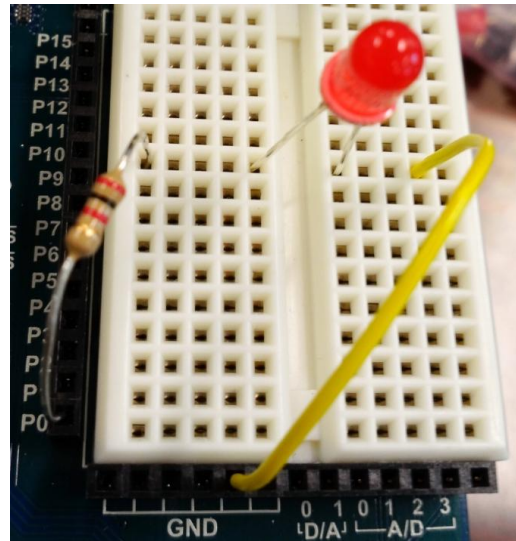
What can we do if we want an LED blink forever? It is not possible for us to keep copying and pasting “`high(0); pause(2000); low(0); pause(2000);`” into our program. The keyword “while” can help us to achieve our goal- repeating the segment infinitely. Keyword “while” accepts a parameter (number) to indicate if the program runs or not. Usually, we use “1” and “0” to represent run and not run respectively, you can also change 1 to any number larger than 1, the program will still work. You can compare the program “BlinkLight(procedure).side” and “Blink Light(while_loop).side”. You’ll notice the former program blink once only but latter one will keeps on blinking.

Steps:

1. Connected resistor from Pin0.
2. Plug a bulb (longer side positive).
3. Run program.

Functions to be customized:

1. `high(pin number)`
Supply power to Pin.
2. `pause(millisecond)`
Retain the former action for assigned millisecond.
3. `low()`
Stop supply power to Pin.



Comparing “Blink Light(procedure).c “ The procedures inside while loop will iterate infinitely.

```
/*Blink Light(procedure).c */          /*Blink Light(while_loop).c */
#include "simpletools.h"                #include "simpletools.h"

int main() {                           int main() {

    high(0);                           while(1) {
    pause(2000);                        high(0);
    low(0);                             pause(2000);
    pause(2000);                       low(0);
                                        pause(2000);
                                        }
}                                       }
```

3b. For loop

A for loop is similar to a while loop. But the difference is that for loop requires users to know exactly how many times they want the segment to run. A for loop has the following format:

```
for(initial value, how many times to loop, increase by some amount every time)
```

In the following program, the buzzer buzzes 3 notes “do”, “re”, “me” using a for loop.

Steps:

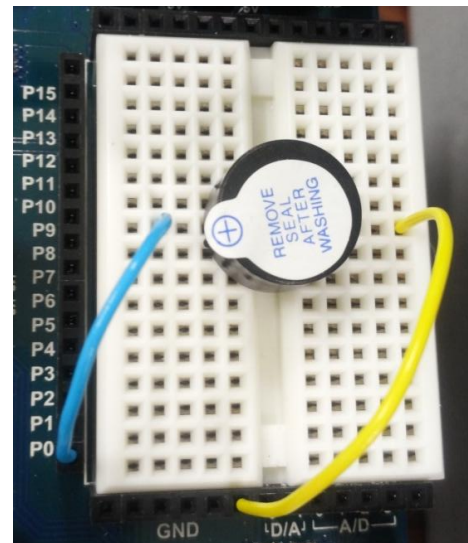
1. Connected buzzer from Pin0.
2. Connect buzzer to ground (GND).
3. Run program.

Functions to be customized:

1. `freqout (pin,duration,frequency)`
Supply power to buzz to make sound.
2. `pause(milliseconds)`
Retain the former action for assigned millisecond.

Variables:

1. `note[]`
Store 3 notes in an array.
2. `i`
Counter variable, monitor the notes.



```
/*buzz(procedure_for_loop).c*/
```

```
#include "simpletools.h"
```

```
int main() {
```

```
    int note[] = {1047,1175,1319} ;
```

```
    int i = 0;
```

```
    for(i=0;i<3;i++) {
```

```
        freqout(0,500,note[i]);
```

```
        pause(68);
```

```
    }
```

```
}
```

`note[]` is an array, you may call it a container variable, storing 3 notes. You can imagine there are 3 variables namely "`note[0]`", "`note[1]`" and "`note[2]`" storing "1047", "1175", "1319" respectively.

This function requires 3 parameters: pin number, buzz duration in millisecond and frequency. Inside for loop, variable "`i`" will increase one every time starting from 0 to 2. `note[i]` can be `note[0] = 1047`, `note[1] = 1175` and `note[2] = 1319`.

Chapter 3

Objective

- ✓ To test some advanced functions with ultrasonic sensor

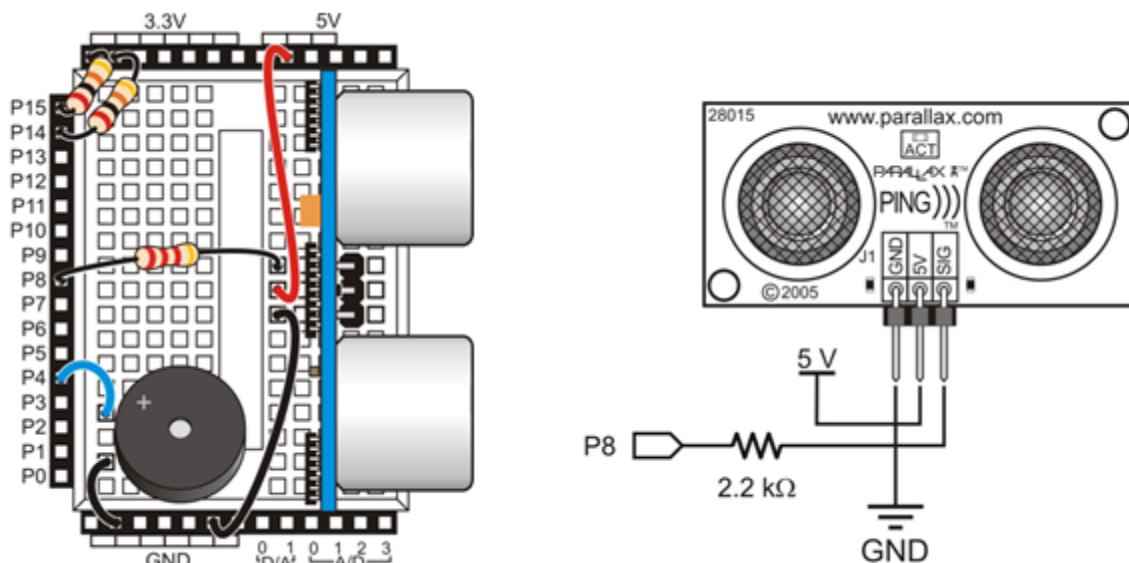
1. testing the ultrasonic sensor

First we need to install the ultrasonic sensor onto the ActivityBot. You will need the following components:

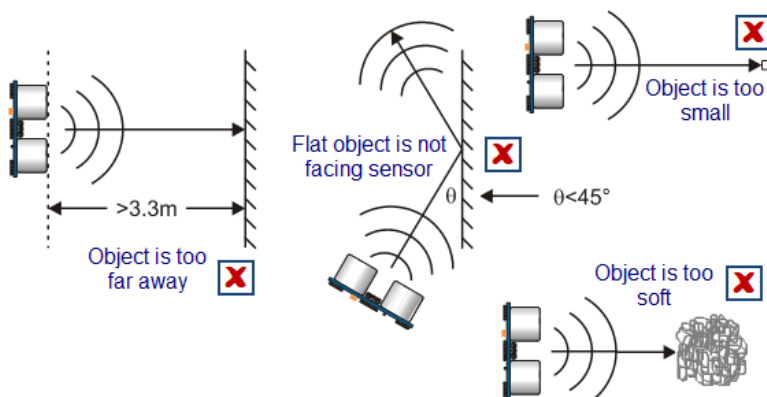
- The PING sensor (ultrasonic)
- 2.2 K ohm resistor (red-red-red colour bands)
- Piezo speaker
- Jumper wires

SWITCH OFF THE ACTIVITYBOT FIRST.

Connect the components as shown in the following.



The PING sensor can detect large-enough objects up to 3.3 meters away. Hard-surface objects work well.



Use the following program to test the PING sensor.

Example: PingSensorTest1.c

```
/*
  Test Ping.c
  Test the PING))) sensor before using it to navigate with the ActivityBot.
*/

#include "simpletools.h"           // Include simpletools header
#include "ping.h"                 // Include ping header

int distance;                     // Declare distance variable

int main()                       // main function
{
  while(1)                       // Repeat indefinitely
  {
    distance = ping_cm(8);        // Get cm distance from Ping)))

    print("%c distance = %d%c cm", // Display distance
          HOME, distance, CLREOL);

    pause(200);                  // Wait 1/5 second
  }
}
```

Test the following:

- Try measuring the distance to your hand, a cup, and a ball. All should be easy to detect.
- Try measuring the distance to something soft, like a stuffed animal or a crumpled piece of fabric. Can the PING))) sensor detect an echo from it?
- Try measuring a wall's distance head-on.
- Try measuring the distance to a wall, approaching at various angles. How far from 90° can the PING))) sensor still see the object?

If you are interested in how ultrasonic sensor works in calculating the distance, you can refer to the following program. Basically, the PING sensor detects the returning echo and use the time taken to work out the distance.

Example: PingSensorTest1.c

```
/*
  Test Ping.c
  Test the PING))) sensor before using it to navigate with the ActivityBot.
*/

#include "simpletools.h"           // Include simpletools header
#include "ping.h"                 // Include ping header

int distance;                     // Declare distance variable

int main()                       // main function
{
  while(1)                       // Repeat indefinitely
```



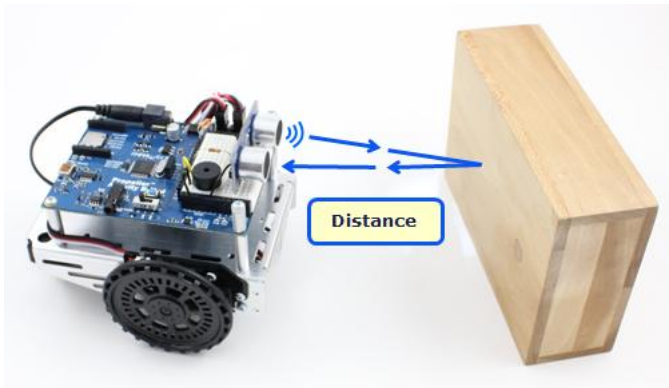
```

{
    distance = ping_cm(8);                // Get cm distance from Ping)))
    print("%c distance = %d%c cm",        // Display distance
          HOME, distance, CLREOL);
    distance = ping(8);
    print("\n echo time = %d%c us", distance, CLREOL);
    pause(200);                          // Wait 1/5 second
}
}

```

2. roaming with the ultrasonic sensor

The following program makes ActivityBot going forward until it receives a measurement from the PING sensor. It means an obstacle is in front. ActivityBot then slow down to a stop and then turn around until the measured distance is more than 20 cm.



Example: PingSensorTest3.c

```

/*
    Detect and Turn from Obstacle.c
    Detect obstacles in the ActivityBot's path, and turn a random direction to avoid them.
*/

#include "simpletools.h"                // Include simpletools header
#include "abdrive.h"                   // Include abdrive header
#include "ping.h"                      // Include ping header

int turn;                             // Navigation variable

int main()                            // main function
{
    drive_setRampStep(10);             // 10 ticks/sec / 20 ms

    drive_ramp(128, 128);              // Forward 2 RPS

    // While distance greater than or equal
    // to 20 cm, wait 5 ms & recheck.
    while(ping_cm(8) >= 20) pause(5);   // Wait until object in range

    drive_ramp(0, 0);                 // Then stop

    // Turn in a random direction

```

```

turn = rand() % 2; // Random val, odd = 1, even = 0

if(turn == 1) // If turn is odd
    drive_speed(64, -64); // rotate right
else // else (if turn is even)
    drive_speed(-64, 64); // rotate left

// Keep turning while object is in view
while(ping_cm(8) < 20); // Turn till object leaves view

drive_ramp(0, 0); // Stop & let program end
}

```

Note that this application uses functions from four libraries, simpletools (pause), abdrive (anything starting with drive), ping (ping_cm), and math (rand).

The above program can be improved so that ActivityBot keeps performing the same trick – moving and avoiding obstacles.

Example: PingSensorTest4.c

```

/*
    Detect and Turn from Obstacle.c
    Detect obstacles in the ActivityBot's path, and turn a random direction to avoid them.
*/

#include "simpletools.h" // Include simpletools header
#include "abdrive.h" // Include abdrive header
#include "ping.h" // Include ping header

int turn; // Navigation variable
int main() // main function
{
    drive_setRampStep(10); // 10 ticks/sec / 20 ms
    while (1) {
        drive_ramp(128, 128); // Forward 2 RPS
        // While distance greater than or equal
        // to 20 cm, wait 5 ms & recheck.
        while(ping_cm(8) >= 20) pause(5); // Wait until object in range
        drive_ramp(0, 0); // Then stop
        // Turn in a random direction
        turn = rand() % 2; // Random val, odd = 1, even = 0

        if(turn == 1) // If turn is odd
            drive_speed(64, -64); // rotate right
        else // else (if turn is even)
            drive_speed(-64, 64); // rotate left
        // Keep turning while object is in view
        while(ping_cm(8) < 20); // Turn till object leaves view
    }
    drive_ramp(0, 0); // Stop & let program end
}

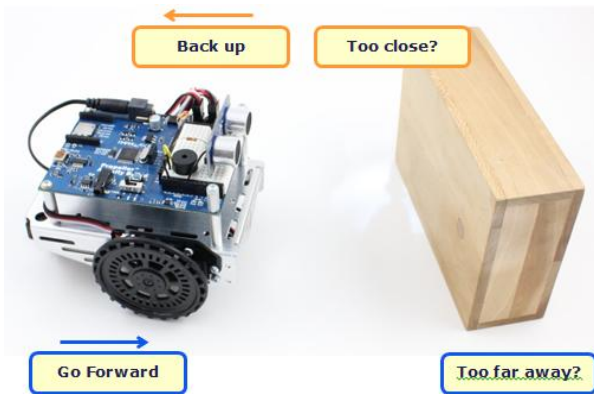
```

You can carry out the following experiments:

- Modify the program so that it turns alternative direction with each obstacle (first time left, and then right, then left again).
- Make ActivityBot stops after finding four obstacles.
- Make ActivityBot explores a maze?

3. guide dog with the ultrasonic sensor

The following program makes ActivityBot following an object.



Example: *PingSensorTest5.c*

```
/*
  Follow with Ping.c
  Maintain a constant distance between ActivityBot and object.
*/

#include "simpletools.h"           // Include simpletools header
#include "abdrive.h"              // Include abdrive header
#include "ping.h"                 // Include ping header

int distance, setPoint, errorVal, kp, speed; // Navigation variables

int main() {                     // main function
  setPoint = 32;                  // Desired cm distance
  kp = -10;                       // Proportional control
  drive_setRampStep(6);           // 7 ticks/sec / 20 ms

  while(1) {                     // main loop
    distance = ping_cm(8);        // Measure distance
    errorVal = setPoint - distance; // Calculate error
    speed = kp * errorVal;         // Calculate correction speed

    if(speed > 128) speed = 128;   // Limit top speed
    if(speed < -128) speed = -128;

    drive_rampStep(speed, speed); // Use result for following
  }
}
```

The program declares five variables: distance, setPoint, errorVal, kp, and speed.

- distance stores the measured distance of an object in front of the ActivityBot.
- setPoint stores the distance the ActivityBot should try to maintain between itself and an object it detects in front of it.
- errorVal stores the difference between the measured distance and the desired setPoint distance.
- kp stores a proportionality constant; errorVal can be multiplied by kp for a speed that will help maintain the setPoint distance.
- speed stores the result of $\text{errorVal} * \text{kp}$.

You can carry out the following experiments:

- Try changing the distance from 32 to 20 (the setPoint variable). ActivityBot will try to maintain a minimal distance of 20, which is more challenging to keep.
- So at the same time, try different kp value, when increased will make ActivityBot go faster in correcting errors.
- You can also decrease drive_setRampStep value which cushion the changes and could ended up slow responsive.

Chapter 4

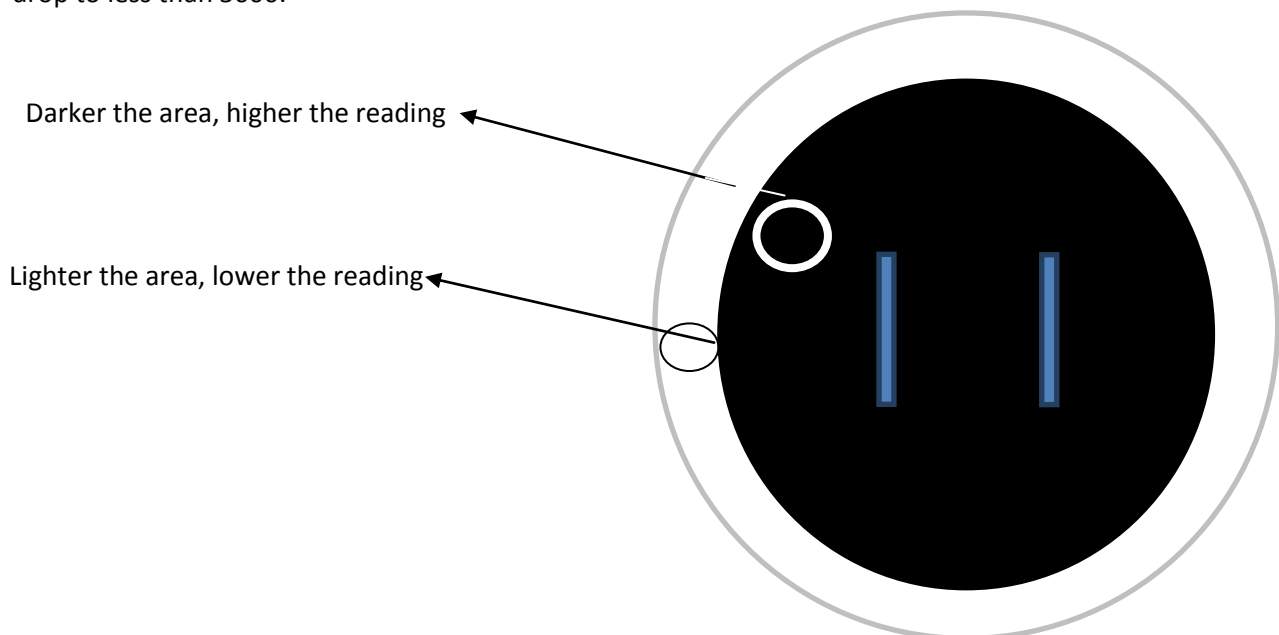
Objective

- ✓ To test some advanced functions (identifying light and dark area, pushing obstacles) using devices

1. Identify black and white area

A circular black and white board will be used in our robot sumo competition. We need to detect the white tape and avoid our robot from entering the area and falling out of the boundary. But how can we detect it? Readings returned by a phototransistor indicate whether the robot enters a bright place, it is the white tape in our case.

When the robot enters a black area, the returned reading increases. If we use a $0.1\mu\text{F}$ capacitor, the reading will be larger than 15000. When it leaves the dark area to a light area, the reading will gradually drop to less than 5000.



Devices:

1. Phototransistor
2. $0.1\mu\text{F}$ capacitor labeled 104 or $0.01\mu\text{F}$ capacitor labeled 103
3. 3 wires
4. 220 ohm resistor (red-red-brown)

Steps:

1. Insert the phototransistor into the LED standoff (the larger of the two pieces) as shown in figure 1.
2. Twist the ending of phototransistor into U-shape as shown in figure 2.
3. Connect resistor from P0 to capacitor.
4. Connect capacitor to the longer leg of phototransistor.
5. Link the shorter leg of phototransistor to capacitor.
6. Connect capacitor to ground.
7. The finished connection should look like the one in figure3-4. You can hang the phototransistor under the robot using 2 wires.



Figure 1



Figure 2

8. Enter the following program and press “Run with Terminal”.

```
/*lightsense.c */  
  
#include "simpletools.h"  
  
int main(){  
  
    while(1){  
  
        high(0);  
  
        pause(1000);  
  
        int t = rc_time(0, 1);  
  
        printf("t = %d\n",t);  
  
    }  
  
}
```

9. Move the robot on a dark and bright board and notice the change.

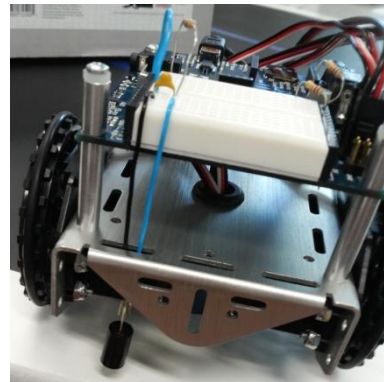


Figure 3

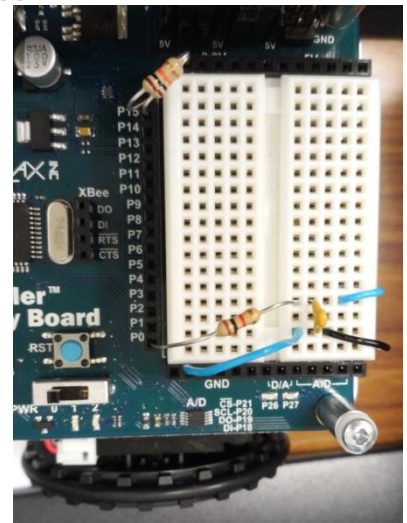
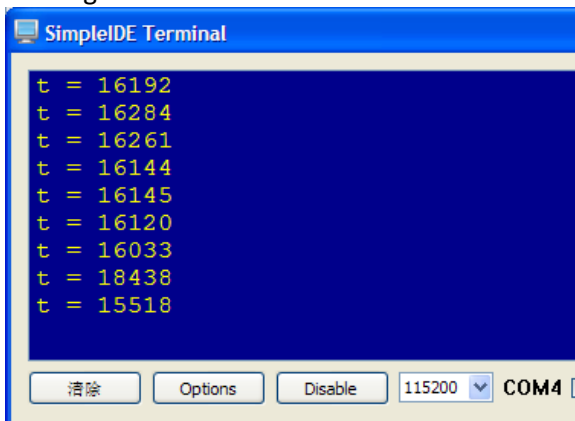


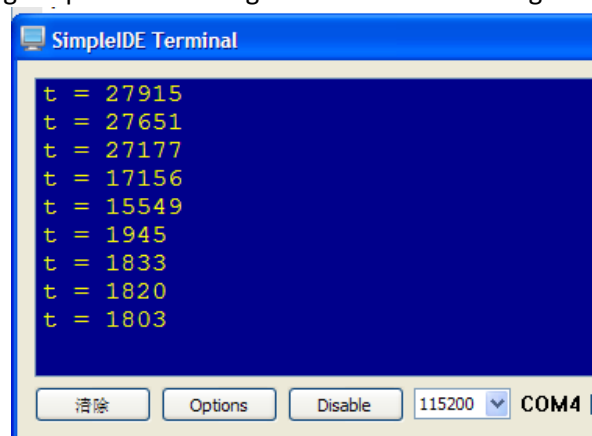
Figure 4

If you have run the program correctly, you should see something like below:

Reading in darker area



Reading drops when moving robot from dark to bright area



2. Detect Obstacle

In a sumo competition, we are going to push our opponent out of the boundary. Before knocking them out, we need to know their position. It seems a good idea to have our robot to spin 36 times (if you remember, a robot will move 36 times, each 10° to finish one circulation), store 36 data and compare which position is closest to it. However, our opponent will move and will not stay at the same place. Therefore, the robot must move forwards once it receives signal.

In this section, we will first learn how to detect obstacle without spinning around. Then, we will program how to move towards obstacle and push it.

2.1 Robot spinning at the same spot and detecting distances between it and obstacles

Devices:

1. Ultrasonic sensor
2. 2 k-ohm resistor (red-black-red)
3. 2 wires

Steps:

1. Connect right servo to Pin12, left to Pin13.
2. Connect a 2 k-ohm resistor (red-black-red) from Pin8 to sensor "SIG".
3. Supply power from 5V to sensor "5V".
4. Connect ground (GND) to sensor "GND".
5. Install sensor.
6. Run the program below.

```
/*returnDistance.c*/
```

```
#include "simpletools.h"
#include "abdrive.h"
#include "ping.h"
```

```
int main(){
    int turn;

    while(1){
        turn = rand() % 2;
```

```
        if(turn == 1)
            drive_goto(17, -17);
        else
            drive_goto(-17, 17);
```

} If variable is odd (1), then the robot turns right.
} If variable is even (0), then the robot turns left.

```
        print("distance = %d\n", ping_cm(8));
```

```
    }
}
```

rand() returns a pseudo random value somewhere from 0 to 2,147,483,648. Each time the returned random value may or may not be the same as the previous value.

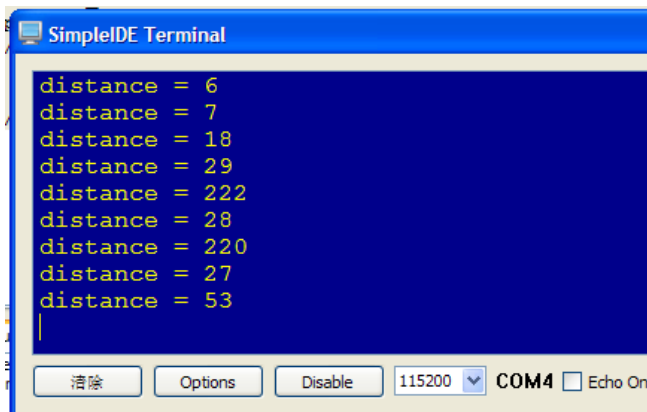
mod 2(%2) is used to check whether the value returned by rand() is divisible by 2 or not. If a value is odd, mod 2 will give 1; otherwise mod 2 will give 0.

E.g. let's say rand() gives us the value "9", "9mod2" will give 1 as 9 can be represented by "2X4+1"

rand() gives us the value "6", "6mod2" will give us 0 as 6 can be represented by "2X3+0"

Display distance

If you have programmed correctly, you should see the robot spinning left and right randomly and the distance detected will be shown on screen.

A screenshot of the SimpleIDE Terminal window. The terminal has a blue background and displays a series of distance sensor readings in yellow text. The readings are: distance = 6, distance = 7, distance = 18, distance = 29, distance = 222, distance = 28, distance = 220, distance = 27, and distance = 53. Below the terminal window, there are several control buttons: '清除' (Clear), 'Options', 'Disable', a baud rate dropdown set to '115200', a COM port dropdown set to 'COM4', and an 'Echo On' checkbox which is currently unchecked.

```
distance = 6
distance = 7
distance = 18
distance = 29
distance = 222
distance = 28
distance = 220
distance = 27
distance = 53
```

Functions to be customized:

1. `drive_goto()`
This function allows the robot to spin certain degree. You'll need to modify the parameter inside `drive_goto()` to the degree you want. Please refer to the degree table provided in Chapter 1.
2. `ping_cm()`
If you install your sensor in different pin, you'll need to change the pin number inside `ping_cm()`.

Functions for references:

1. `turn = rand() % 2;`
The `rand()` function will randomly pick a number from the range between 1 and 2,147,483,648. So that when the number mod 2, it will give either 1 or 0 to turn.

2.2 Robot moving forward when detecting obstacles inside certain distance

We have already learnt how to find the distance between two objects. Now, we will learn how to approach the opponent once it enters certain distance. Modify your previous program in 2.1.

Steps:

1. Modify your code to:

```
#include "simpletools.h"
#include "abdrive.h"
#include "ping.h"
```

```
int main(){
    int turn;

    while(1){
        turn = rand() % 2;

        if(turn == 1)
            drive_goto(17, -17);
        else
            drive_goto(-17, 17);
```

Add this
segment to
your
program

{
object
}

```
while(ping_cm(8) <= 40){
    drive_setRampStep(10);
    drive_ramp(14, 14);
}
```

When opponent is found within 40cm.

You can modify 40 to any distance

→ set maximum speed, 10 ticks per sec

→ gradually increase speed and move towards

2. Load the program into your robot.
3. Disconnect the cable and run your robot on a smooth ground.

Functions to be customized:

1. `drive_setRampStep()`

This function sets the number of ticks per second that the speed is allowed to change for every 50th of a second (maximum speed). You can increase the number of ticks per seconds, so that the robot will move faster and vice versa.

2. `drive_ramp(,)`

This function ramps the robot up to full speed, in steps of the number of ticks defined in `drive_setRampStep()` ticks per second. E.g. With a ramp step of 10 and 14 as full speed, it takes $((14/10)*1/50) = 0.028$ seconds to ramp up to full speed. You can modify the parameter so that the acceleration becomes more or less obvious.

3. Conclusion

In these three lessons, we have learnt how to compile and load programs using SimpleIDE, some simple programming concepts (procedures, condition and iteration) and functions to run the ActivityBot.

In the first lesson, we learnt to construct, test and calibrate a Parallax ActivityBot. Then, we tried some easy functions and calculations to make the ActivityBot moving forward and backward for certain distance and turning in different degree, making it move in circle, triangle and other shapes.

In the second lesson, we learnt some basic programming techniques. We first tested a procedure program using an LED light bulb and made it blink. Later, we tried some conditions on “robot will stop 5 cm before an obstacle” and “an LED light blink when a phototransistor detects no light”. Then, we tested `while` loop using an LED light bulb. We compared the `while` loop program to the ordinary procedure program, we noticed that if we put the procedure program into a `while` loop, the LED light will blink infinitely. Finally, we learnt `for` loop by producing 3 notes from a buzzer. An array can be used to make the whole program looks more neat and simple.

In the final lesson, we tried to combine different programming techniques and functions to make our ActivityBot moved the way we wanted. We learnt how to detect black and white area and avoid the robot from falling out of boundary. We also examined a program about detecting an obstacle and moving to it once found.

After the lessons, you should understand more about programming- variable, function, procedure, condition and iteration. Programming is just like your ActivityBot, you need to put every little device together step by step to make it work.