Simulator Details

If you guys are having trouble testing the bot again and again, you can now use a simulator to test your algorithm and image processing before testing it on a real bot. Unfortunately, the simulator provided is written in C++ using opencv, so participants using matlab or other language have to write their own code. The code is short and just uses some mathematical equations, so it should be easy to understand and rewrite. Still, if you have any problem, feel free to contact us. Participants using C can use it and compile their code in C++ instead of C, cause Code written in C is compatible with C++.

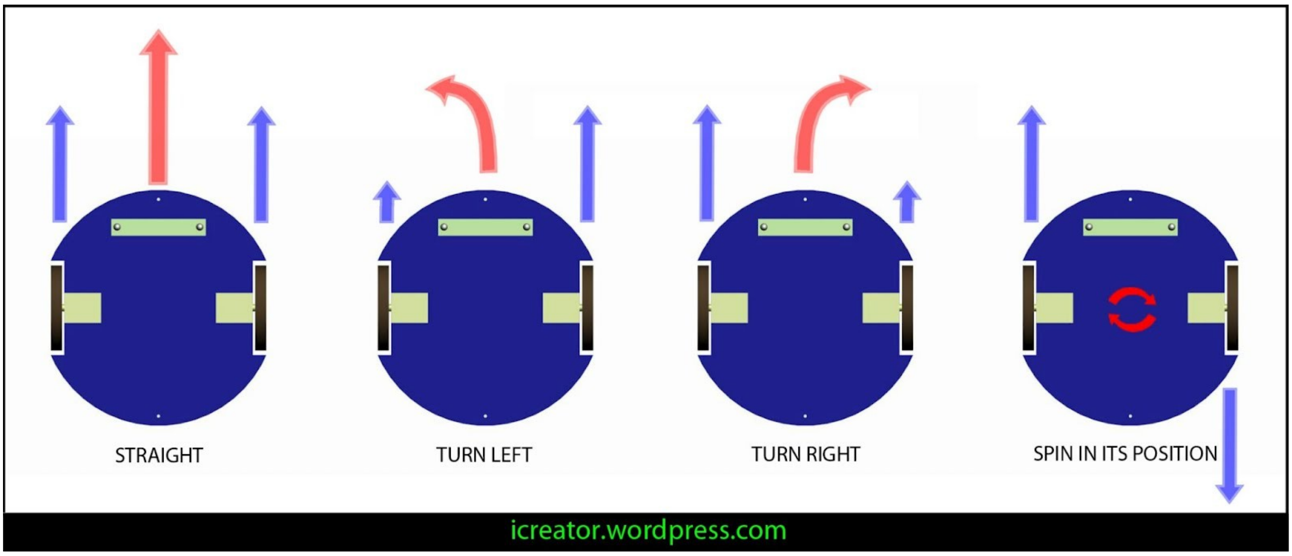
The classes of simulator uses OpenCV and Mat objects to draw the 'virtual bot', so can be used only if other code uses opencv as well.

Note: the file uses math.h library, so don't forget to include and link it..

Note: All co-ordinates in the code below follows image processing co-ordinate system, defined in code snippets given on Quark website (green-mars page).

Simulator

The class Differential is a simulation of a 2 wheel differential drive bot. A differential drive mechanism used in wheeled robotics, just says that to make your bot turn, just change speeds of your wheels..
Rotating both wheels in opposite direction makes your bot rotate about its center.
For left turn, make left wheel go slower(or reverse) and right wheel go faster.
For right turn, make right wheel go slower(or reverse) and left wheel go faster.
For straight, both wheels must be rotating at same speed.



icreator.wordpress.com

So, since now we know that we can come back to simulation.
First of all, to use it download the file from the site and copy it to same location as your code.
And include the file in code..

```
#include"differential"
```

Then Construct its object..

```
Differential diff( width, length); //width and length are in pixels
```

then set its wheel speeds and bot's angle theta..

```
diff.vl =left wheel speed;
diff.vr =right wheel speed;

diff.theta = angle;
```

where both speeds are integer between -255 to 255;
and angle follows image processing co-ordinate with -PI/2 for up, PI/2 for down,0 for right,PI for left.(Co-ordinate explained in code snippets uploaded on website)
If you are using arduino for your bot, you can later send the same speed to arduino to be given to analogWrite function, which takes a integer between 0 to 255 as parameter.

Angle can be calculated using markers or tags mounted on bot (for virtual tags scroll down) (for information on tags see code snippets on website)

Now that you have set these, just call

```
diff.update(display,h1, s1,v1,h2,s2,v2 );
```

where display is a opencv Mat object or image (in HSV format) you want to display bot on, h1,s1,v1, and h2,s2,v2 are hsv values to be used to draw the bot.

Virtual Markers

You might also need to attach markers on this virtual bot. For that I have provide fourtags class/file that (despite its name :P) attaches 5 tags on your virtual bot, one on center and four on all four directions.

To use it, first copy the file given on website, to the same location as your code, and include the file in your code.

```
#include"fourtags"
```

Then construct its object,

```
FourTags tags(width,length);  //width and length are in pixels
```

then, set its position and angle theta,

```
tags.centerx=diff.x+offset*cos(diff.theta);
tags.centery=diff.y+offset*sin(diff.theta);

tags.theta=diff.theta;
```

where diff is your differential simulation object, and offset is the offset(in pixels) you require for center tag from center of virtual bot.

(don't forget to set values for diff first, as shown above)

Then, time to draw it..

```
tags.draw(display,h1,h2,h3,h4,h5);
```

where diplay is your opencv Mat object or image(in HSV) you want to draw on.
And h1,h2,h3,h4,h5 are 5 hue values to be used to draw 5 tags..

If you want to avoid image processing to find centers of indivisual tags, you can use:
tags.getBack() , tags.getFront() , tags.getLeft() , tags.getRight() , tags.getCenter()

All these functions returns a object of struct coordinate, which has 2 float objects:x and y;
So, the position of back tag would be:

```
x= tags.getBack().x;
y= tags.getBack().y;
```

(in image processing Co-ordinate system (y inverted));

Thats all about simulation.
For any help or queries, mail me at greenmars@bits-quark.org.
For other contact details or details about the event, please visit http://www.bits-quark.org/2013/greenmars