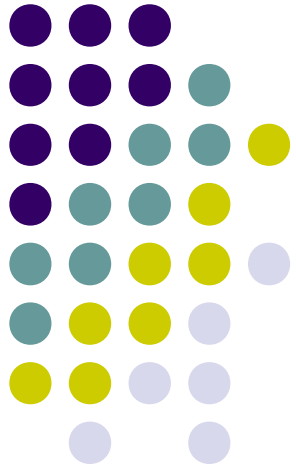
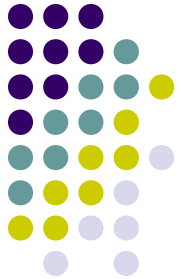
Blob following with obstacle avoidance

Presentation by Sharice Handa
for Professor Francesco Bullo
June 12, 2009



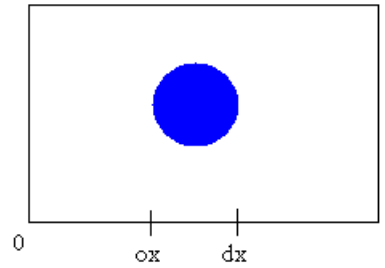
Goals (Fall 08)

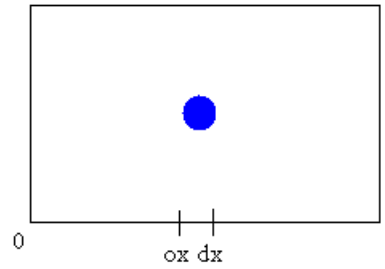


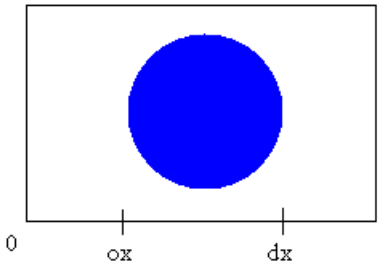
- Learn C++ Programming Language
- Learn Player/Stage
- Write code to use camera and blob finder to follow a blob

Blob Follow: Finding change in distance

- Blob moves closer or farther away
  - Camera shows:
    - Blob at desired distance
      - Blobfinder driver gives ox and dx values
    - Blob has moved farther away
      - (Blob in camera becomes smaller)
    - Blob has moved closer
      - (Blob in camera becomes larger)



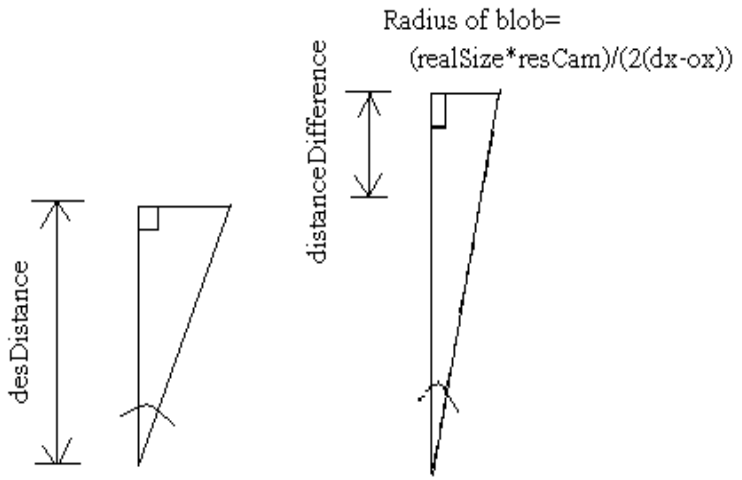




Blob Follow: Finding change in distance

```
double distanceDifference=-desDistance+((realSize*resCam/2)/(dx-ox)/tan(fov/2));
```

$$\text{double distanceDifference} = \frac{(\text{realSize} * \frac{\text{resCam}}{2})}{\frac{(dx-ox)}{\tan(\frac{fov}{2})}} - \text{desDistance}$$



- desDistance is the user specified distance which the robot should stay away from the blob.
- realSize is the user defined real size of the blob.
- ResCam is the resolution of the camera (number of pixels)
- (ox-dx) is the diameter of the blob from blobfinder data of camera.
- fov is the field of view of the camera (angle)

tan(fov/2) = Radius of blob / (distanceDifference + desDistance)

Blob Follow: Finding change in distance

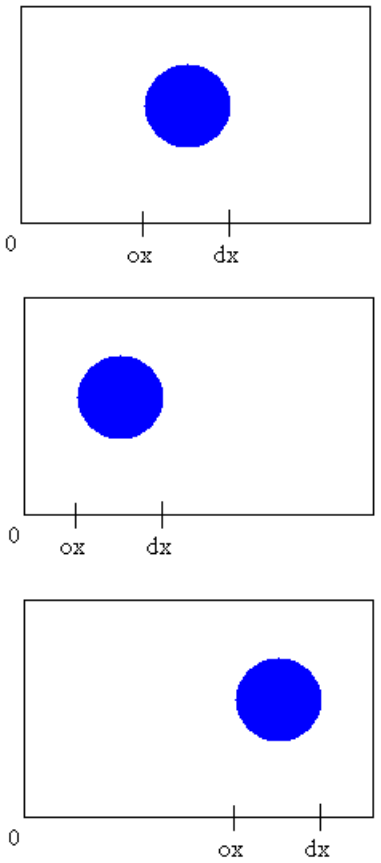
```
//change of distance
if (abs(distanceDifference)<=.1)
{
    distToGoal1=0;
}

else if (abs(distanceDifference)>=.1)
{
    distToGoal1=distanceDifference;
}
```

- If calculated distanceDifference is less than .1m, robot dose not move.
- If distanceDifference is more than .1m, robot drives to that distanceDiference.

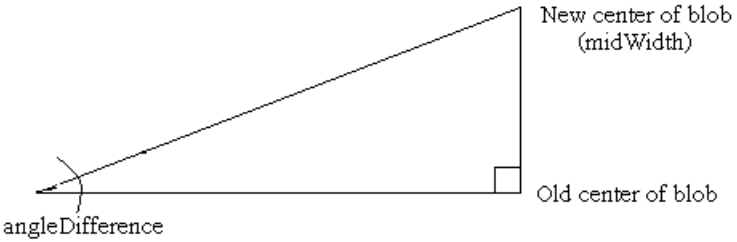
Blob Follow: Finding change in angle

- Blob moves to the left or right
  - Camera shows:
    - Blob is centered
    - Blob has moved to the left
      - (Blob in camera offset to left)
    - Blob has moved to the right
      - (Blob in camera offset to right)



Blob Follow: Finding change in angle

```
double angleDifference=(resCam/2-midWidth)*fov/resCam;
        //change of angle
        desiredHeading1=angleDifference;
```



$$\text{double angleDifference} = \left(\frac{\text{resCam}}{2} - \text{midWidth}\right) * \frac{\text{fov}}{\text{resCam}}$$

- ResCam is the resolution of the camera (number of pixels)
- midWidth is (ox-dx)/2 is the middle of the blob (widthwise)
- fov is the field of view of the camera (angle)
  - fov/resCam is the number of degrees per pixel

Blob Follow: Demo Video

combined change of distance and angle

Play BlobFollowDemo Video

Notice:

- The red robot, follows the blue blob.
- A visual representation of the robot's camera is present.
- The robot keeps a user defined distance away from the blob.
- The robot tries to keep the blob at the center of it's view.
- When the robot loses sight of the blob continues at last heading and angle.
- No obstacle avoidance.

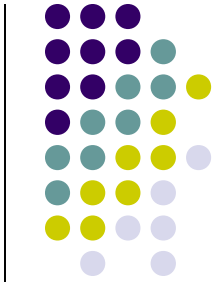
Blob Follow: combined change of distance and angle

- Problems thus far:
  - If robot loses sight of the blob it will continue at last sighted angle and distance until blob is found again. Potentially crashing into walls.
  - Robot cannot follow blob around corners. Hits corner.
- Solution:
  - Add code to make robot stop movement if loses sight of blob.

```
if (ox==0 && dx==0)
{
    cout<<"No data"<<endl;
    desiredHeading=0;
    pp.SetSpeed(0,0);
   continue;
}
```

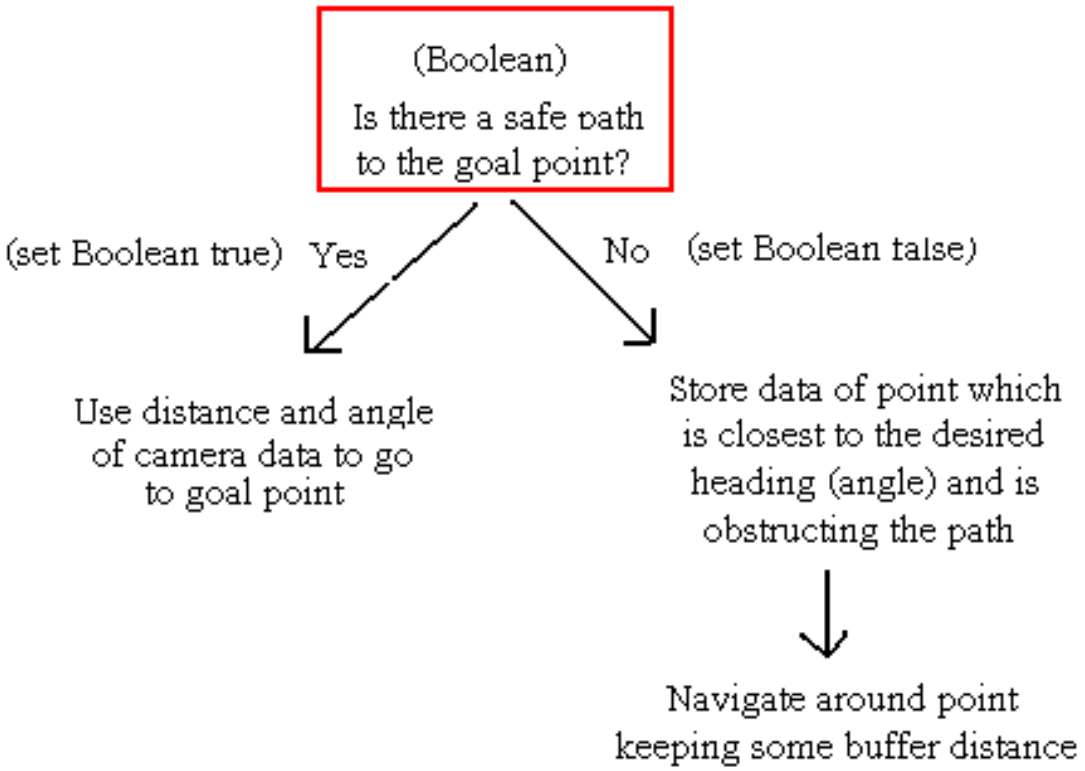
  - Integrate laser proxy data to help robot navigate around corners and obstacles while still following blob.

Goals (Winter 08)



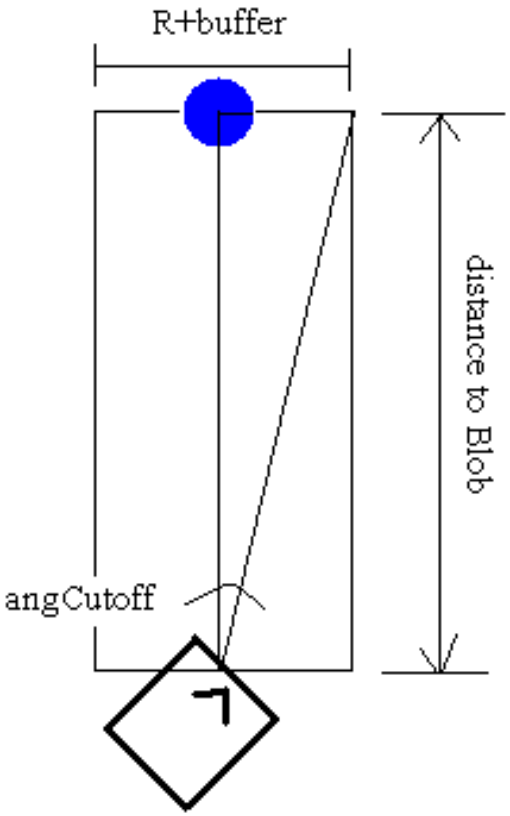
- Integrate use of laser data to avoid obstacles while following blob

Blob Follow: obstacle avoidance



- Create Boolean.
- loops through laser data points.
- Boolean is true if the path from the robot to its desired new spot is clear .
- Boolean is false if the path is not clear.

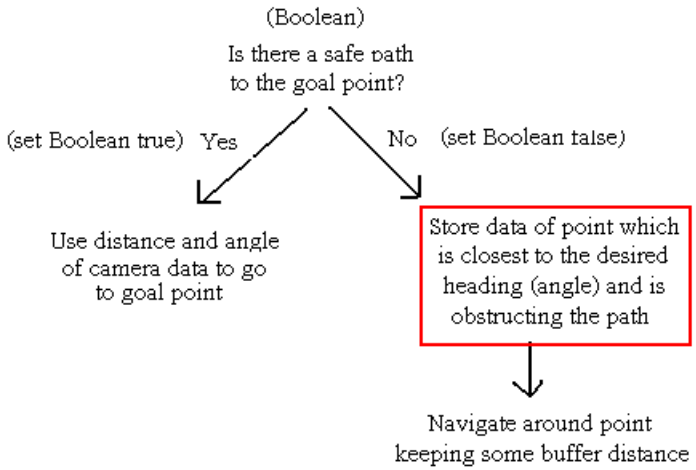
Blob Follow: obstacle avoidance




```
bool SafePathToGoal=true;
double angCutoff=atan2(R+buffer,distanceToBlob);
 // loop through laser distances
 for( int j = 0; j < count; j++ )
 {
    double angleJ=angRes*(j-count/2)-angleDifference;
    double drectJ=0;
 }

if (fabs(angleJ)<angCutoff)
 {
     drectJ=fabs(distanceToBlob/cos(angleJ));
  }
else if (fabs(angleJ) < M_PI/2.0)
 {
    drectJ=fabs((R+buffer)/sin(angleJ));
  }
```

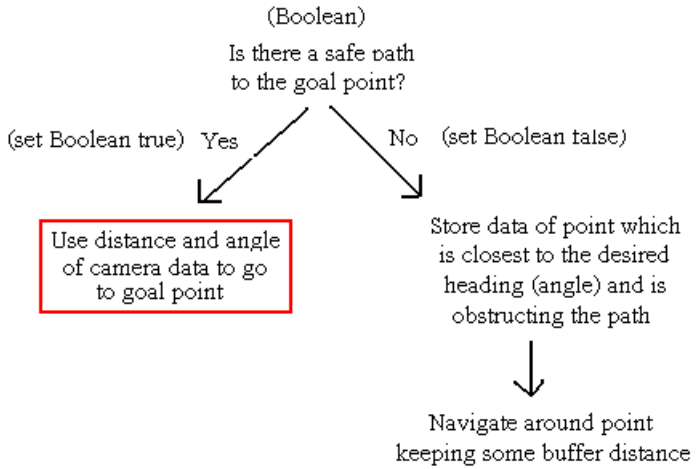
Blob Follow: obstacle avoidance




```
if (lp[j] < drectJ)
{
   SafePathToGoal=false;
   if (fabs(angleJ)<fabs(angleTrouble) )
   {
      trouble=j;
      angleTrouble=angleJ;
      distTrouble=lp[j];
   }
}
```

- Loop through laser data
  - stores the data point which is: closest to the centerline between the robot to the blob and is obstructing path.

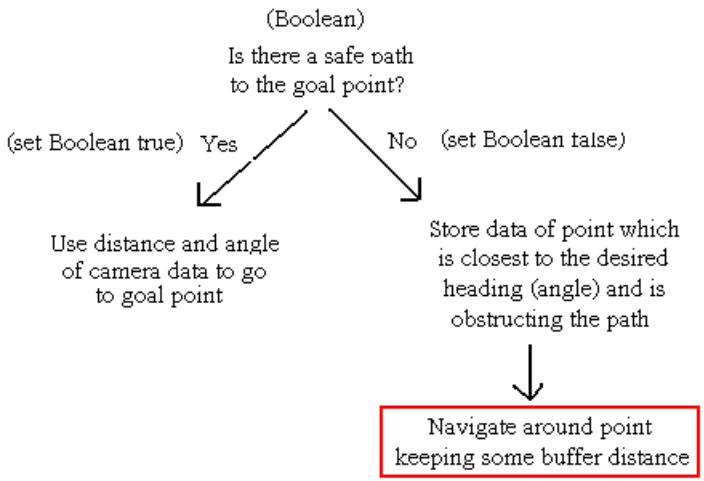
Blob Follow: obstacle avoidance




```
if (SafePathToGoal)
{
    cout<<"Safe Path"<<endl;
    desiredHeading=desiredHeading1;
    distToGoal=distToGoal1;
}
```

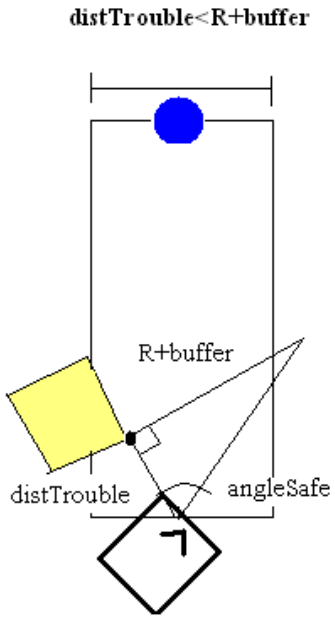
- If there is a safe path to the goal➔Go

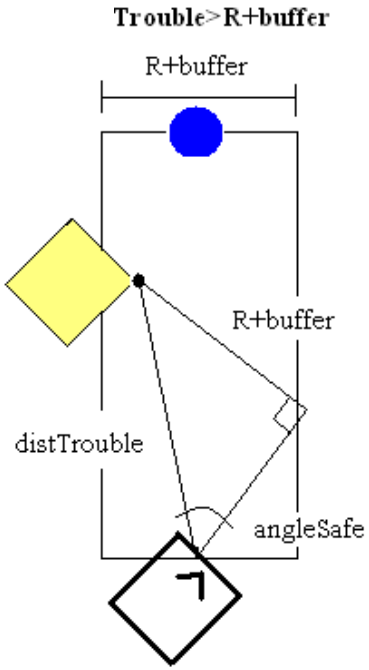
Blob Follow: obstacle avoidance



- To navigate around obstacle point:

Blob Follow: obstacle avoidance



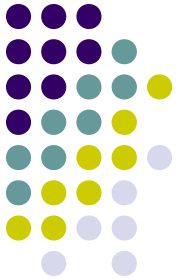



```
else
 {
        if (distTrouble<(R+buffer))
         {
            angleSafe=(M_PI/2)-atan(distTrouble/(R+buffer));
         }
         else
         {
            angleSafe=asin((R+buffer)/distTrouble);
          }

         if (angleTrouble <= 0)
          {
            desiredHeading = angleTrouble+angleSafe;
            distToGoal=distToGoal1;
           }
          else
          {
            desiredHeading = angleTrouble-angleSafe;
            distToGoal=distToGoal1;
           }
}
```

Blob Follow: Demo Video

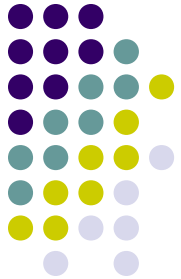
blob following with obstacle avoidance



Play BlobFollowDemo2 Video

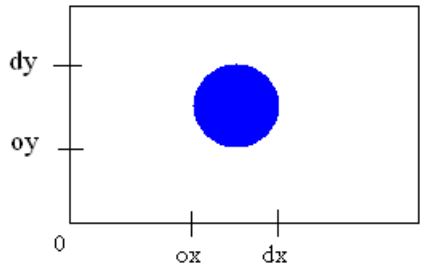
- Robot moves forward, backwards, left and right with blob.
- Maneuvers around corners to follow blob.
- When loses sight of blob (no cam data) stops completely
- Can follow the blob along a wall without crashing.

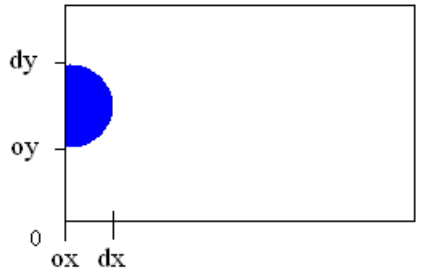
Goals (Spring 09)



- Get more accurate camera data.
- Robot go to last seen location of blob if loses sight of the blob.
- Fix bug in program where sometimes robot will go in opposite direction of blob.

Blob Follow: improving camera data.





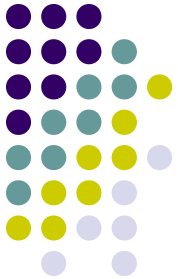
- Instead of using ox and dx values (from x-axis) use oy and dy (from y-axis).
- When only half of the blob is visible, y-axis values of the blob more accurately determines the distance from robot to blob.

Blob Follow: adding memory of last location of blob.

```
yGoal = ( distanceDifference + desDistance ) * sin( theta + angleDifference )+ yPos;
xGoal = ( distanceDifference + desDistance ) * cos( theta+angleDifference ) + xPos;
```

- Initialize the position proxy of the robot.
- At end of defining distanceDifference and angleDifference, store values yGoal and xGoal for later use.
- xGoal and yGoal are the x and y positions of where the blob was last sighted.

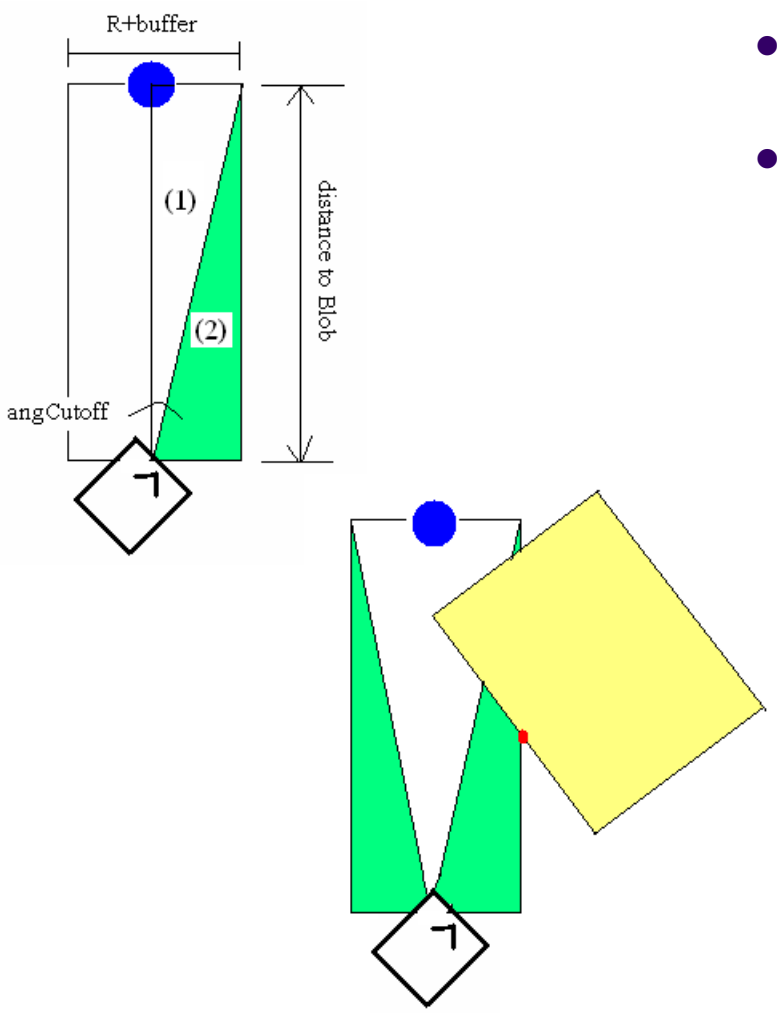
Blob Follow: adding memory of last location of blob.



```
distanceToBlob = sqrt( pow( xGoal - xPos, 2 ) + pow( yGoal - yPos, 2 ) );
angleDifference = normalize( atan2( yGoal - yPos, xGoal – xPos ) – theta );
```

- Instead of telling robot to stop when it does not have camera data, used stored xGoal, yGoal, and position proxy data to determine where the robot wishes to travel.
- When gets to point it last saw the blob, robot circles, waiting until it can see the blob (has camera data) again.

Blob Follow: fixing Boolean.



- Problem:
  - Blob it self was seen as an obstacle.
- Why?
  - The camera data still had errors.
  - To the camera the blob appeared smaller, so the estimated distanceToBlob was larger.
  - Thus the blob was within the checking triangles and the laser data read the blob as an obstacle.

Solution:

- Instead of checking weather triangles (1) and (2) are clear of obstacles, only check (2). Parts highlighted in green.
- If there is an obstacle (in yellow), robot will use the closest obstacle point as it's point of navigating around (in red).
- Same trig formulas as before are used to navigate around obstacle point.

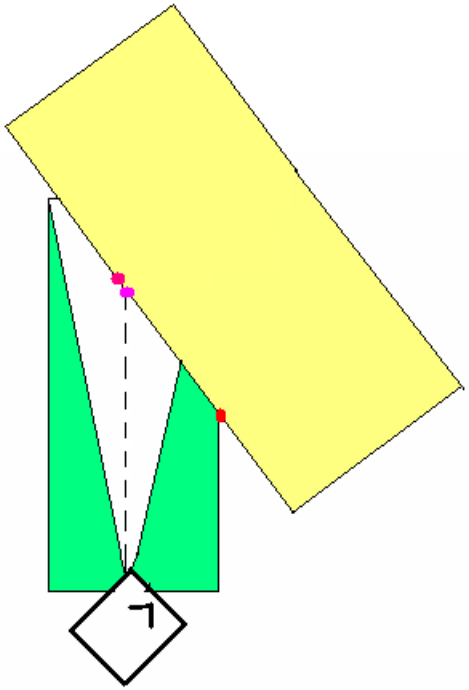
Blob Follow: fixing Boolean.

```
for( int j = 0; j < count; j++ )
    {
    double angleJ=angRes*(j-count/2);
    double drectJ=0;
    if (fabs(angleJ)>angCutoff && fabs(angleJ) < M_PI/2.0)
    {
        drectJ=fabs((R+buffer)/sin(angleJ));
        if (lp[j]<distTrouble && lp[j] < drectJ)
        {
            SafePathToGoal=false;
            distTrouble=lp[j];
            angleTrouble=angleJ;
            trouble=j;
        }
    }
}
```

- Only checks triangle (2)
- Stores smallest laser distance in triangle (2) and that point's angle.
- Uses this to determine safe deflection. (angleSafe)

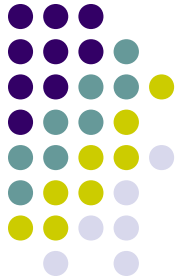
Blob Follow: fixing Boolean.

- Using the closest obstacle point
  - In old program the robot would chose point closest to centerline to use as obstacle point.
    - switch between light and dark pink points being the obstacle point.
    - Would crash into wall.
  - In new program this is no longer a problem.



Blob Follow: Demo Video

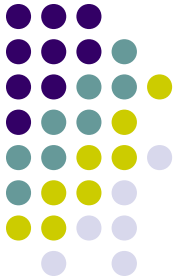
Basic operations



- See Blob_Follow_Final1
  - Basic operations shown, forward, backward, left and right still work.

Blob Follow: Demo Video

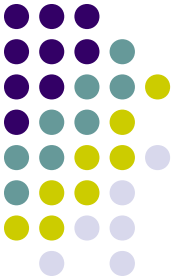
Around a corner



- See Blob_Follow_Final2
  - Robot follows blob around a corner.

Blob Follow: Demo Video

Avoiding a constant wall



- See Blob_Follow_Final3
  - Robot follows blob while avoiding wall.

Blob Follow: Demo Video

Robot continues to last known location of blob

- See Blob_Follow_Final4
  - The robot follows the blob around a corner.
  - Half-way through, the simulation is paused and the blob is moved out of view.
  - The robot continues to the last known location of the blob and turns in circles awaiting new camera data.

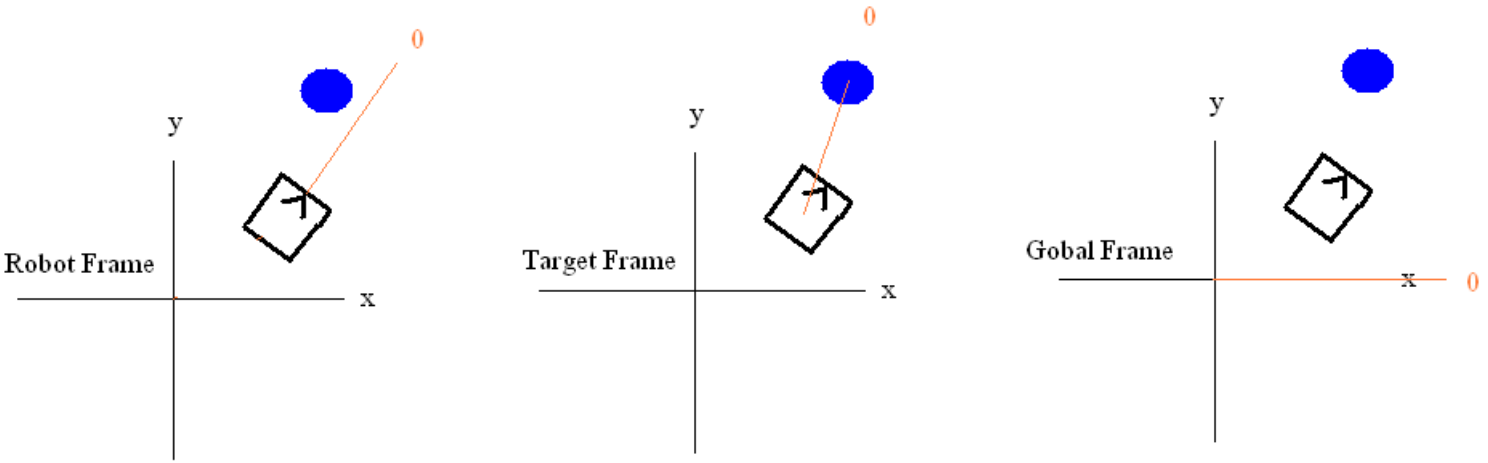
Blob Follow: Demo Video

Blob will continue to seek

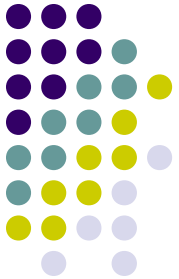
- See Blob_Follow_Final5
  - Robot follows blob.
  - Half-way through, the simulation is paused and the blob is moved out of view.
  - The robot continues to the last known location of the blob and turns in circles awaiting new camera data.
  - When robot is moved, it finds it's way back to the location it last saw the blob with obstacle avoidance.

Challenges

- Debugging program.
  - Used Graphics2DProxy
    - predict where robot thought blob was.
      - Determined that camera data was not accurate, over estimated where blob was.
    - predict where robot thought obstacle was.
- Determining relative frames of reference.
  - Robot frame vs. target frame vs. global frame.



Future Goals



- Make robot less jittery when going around corners
- Implement on hardware.