'YOU CAN RUN, BUT YOU CANNOT HIDE': TRACKING OBJECTS THAT LEAVE THE FIELD-OF-VIEW

JACOB HAUENSTEIN

Computer Science Department, University of Alabama in Huntsville
Huntsville, AL 35899, USA
jhauenst@cs.uah.edu

RAMAZAN TINAZTEPE

Department of Mathematics, Alabama A&M University
Normal, AL 35762, USA
ramazan.tinaztepe@aamu.edu

RAMAZAN S. AYGUN

Computer Science Department, University of Alabama in Huntsville
Huntsville, AL 35899, USA
raygun@cs.uah.edu
http://www.cs.uah.edu/~raygun

Received (Day Month Year)
Revised (Day Month Year)
Communicated by (xxxxxxx)

In the past, there has been significant research on tracking objects based on features that characterize objects. However, once the features to detect the object are not available from the position of a tracker, it is not clear how to track the object. Our goal in this paper is to track an object even though the object leaves the field-of-view. In this paper, we firstly describe the problems related to this type of tracking. We then explain our approach that has two phases: real object tracking and virtual object tracking. We mostly focus on virtual object tracking that requires reorientation, distance estimation, traveling the computed distance, estimating the direction for turn, and turning into the direction. We show our results by building a robot that achieves these.

Keywords: Object tracking.

1. INTRODUCTION

Object tracking has many applications including motion-based recognition, automated surveillance, video indexing, human-robot interaction,[22] traffic monitoring, and vehicle navigation.[1] Object tracking is mainly composed of two phases: the extraction of identifying features of the object and tracking these features. The current research assumes a) the availability of these features continuously or b) if these features are occluded, they will be eventually available.

A major problem yet to be resolved in object tracking is the leave-of-field-of-view. Even a recent survey[1] on object tracking does not provide related research in this area since previous approaches usually assume that the object will eventually appear. The concepts of "leaving the field of view" and "occlusion" are very similar and in some contexts they could even be considered the same. Tao et al.[20] distinguish object occlusion and object disappearance. Object disappearance occurs if a) a moving object moves out of the scene, b) a stationary object is not visible due to camera movement, and c) an occluded object leaves the scene or no motion is detected around it. Object occlusion occurs if there is no motion around the object and template matching is not successful. In our case, the object may move out of the scene due to object occlusion as the camera system tries to relocate itself to reach the object.

1.1. *Related Work*

We classify the research on object tracking into based on the camera movement as *static* and *mobile* (Fig. 1). With static camera, object tracking research targets multiple object tracking and tracking in the presence of object occlusion. For example, while Lanz[16] proposes a Bayesian approach for tracking, Huang et al.[14] use a customized genetic algorithm for region tracking, adaptive appearance models, spatial distributions and inter-occlusion for object tracking. They use a static camera and object should be visible again to track that object. These methods do not track objects that leave the field-of-view of the static camera.

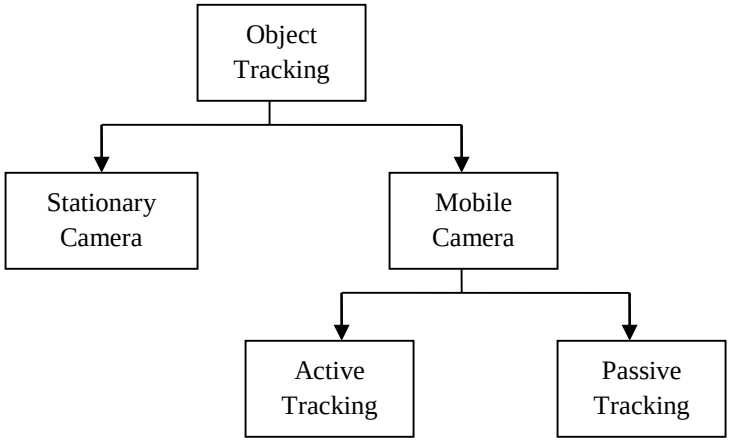


Fig. 1. Classification of tracking

Mobile camera tracking can be classified with respect to the physical tracking of the object and limitations of the camera movement (Fig. 1). In mobile camera tracking, the goal is to track objects as the camera moves. Mobile camera tracking can further be classified as *passive* and *active* in terms of physical tracking of the object. In passive tracking, the camera system tracks moving objects in the field-of-view without physically tracking those objects. For example, Leibe et al.,[15] Pellegrini et al.,[17] and Ess et al.[18] provide methods of passive tracking from mobile platforms. They do not aim to track objects that disappear. In active tracking, the camera system tracks the objects physically wherever they go. Jung et al.[20] provide an object tracking algorithm without geometric constraints from a mobile robot using particle clustering algorithm. Their method does not address object occlusion or disappearance of objects. However, they state that sometimes objects might be occluded. Since they do not have a specific algorithm for disappearance, it is likely that their robot fails if the object disappears. In Ref. 2, multiple sensors including a camera, two microphones, and laser range finder are used to track a person. Lin et al.[3] also propose a tracking method based on similar sensors. Their algorithm, in addition, tries to avoid the obstacles during tracking as in Ref. 4. The use of multiple sensors is important to ascertain that at least some feature about the person is available. In Ref. 5, a robotic dog is proposed to track a person in a room by centralizing the object to be tracked using salient features and color histograms. Elkvall et al.[6] propose a method to detect, approach, and grasp objects in domestic environments. In Ref. 7, the control strategy to track a moving object is composed of two phases: heading regulation and tracking. When the object is not in the line of sight, heading regulation is needed. They explain their methodology with different trajectories of a moving object. Tovar et al.[8] provide an optimal navigation environment for simple connected environment. Their algorithm enables exploring and searching a static object in an enclosed environment. In Ref. 9, the robot is equipped with tactile and position sensors. The goal is to reach a target object in an environment with obstacles. Chen et al.[10] provide a method of object tracking while avoiding obstacles. The object is always sensible through the sensors of the robot even in the presence of obstacles.

In general, the proposed algorithms require the availability of features to be tracked or the proposed techniques assume that features of the object eventually become available. There are also algorithms for

exploration of the environment which can be further used for searching static object. We have not found any strategy for tracking objects that leave the field of view of the robot.

1.2. *Our Approach*

The visibility of the object provides constant information which can be used to command movement in the proper direction. If the object disappears around a corner of a hallway or behind a garbage bin, tracking the object becomes more difficult. It is no longer possible to gather information about necessary movements by examining the object features. The only information available is information that was collected while the object was visible. Thus, in order to track an invisible object, it is necessary to collect and utilize effectively the information that is available when the object was visible. In this project, we propose to develop a robot that is able to track an object that leaves the field-of-view. The major issue in this type of object tracking is the unavailability of features to be tracked after the object becomes occluded. In this paper, we provide an *active mobile tracking system* where our robot physically relocates itself to reach the object. To resolve this problem, we divide the tracking phase into two phases: real object tracking and virtual object tracking. In the real-object tracking phase, the robot tracks the object based on the available features. When the object leaves the field-of-view, virtual object tracking phase starts. In this phase, the idea is to track the last appearance of the object. When the object is visible, real object tracking phase starts again. Our research has the following properties:

(1) The camera system is mounted on a mobile platform (i.e., this is not tracking from stationary cameras)
(2) Our tracking system is an active tracking where our robot moves itself to reach the object. Other research tracking with mobile platform usually tries to track objects as long as they are in the scene. Those methods do not react if the object leaves the field-of-view.
(3) Our tracking system continues to track the object even though the object may disappear.

We should note that our goal is not to design an advanced robot with numerous powerful sensors. We designed a robot with a single vision sensor where we can observe the problem and then test our algorithm on this robot. Our goal is not to develop a low-level feature extraction method for object tracking. Object segmentation is not part of the proposed research. Our goal is to develop an algorithm for tracking an object if the features of the object are not available.

This paper is organized as follows. The following section describes the problem and explains our methodology. Section 3 explains the virtual object tracking phase. The mathematical modeling is covered in Section 4. Section 5 discusses our experiments. The last section concludes our paper.

2. OUR METHODOLOGY

The problem that must be solved is the case where the object being tracked disappears, perhaps around the corner of a hallway or behind a garbage bin. Clearly, this case requires a different method than those cases where the object is visible. In this case, the current status or location of the object cannot be used to make decisions about what movements to make. The only information available to use is that information acquired while the object was visible.

There are no known strategies to handle if the target features cannot be detected anymore. Actually, unavailability of these features might mean that the target object may be occluded by another object or it may not be in FOV of sensors. In such cases, the tracker robot just stands still since no information can be identified to track the target object. Fig. 2 (top row) shows sample scenarios where object tracking fails. Fig. 2 (bottom row) shows how tracking can be achieved for the corresponding scenarios.

In order to effectively relocate the object after it disappears, it is necessary to know in which direction the object was moving, and approximately how far away the object was. This information can be used to drive the robot the appropriate distance forward and rotate in the direction the object exited in order to, hopefully, place the object once again in view of the camera and continue tracking normally. However, there are a number of problems which must be overcome in order to acquire or effectively utilize this information.

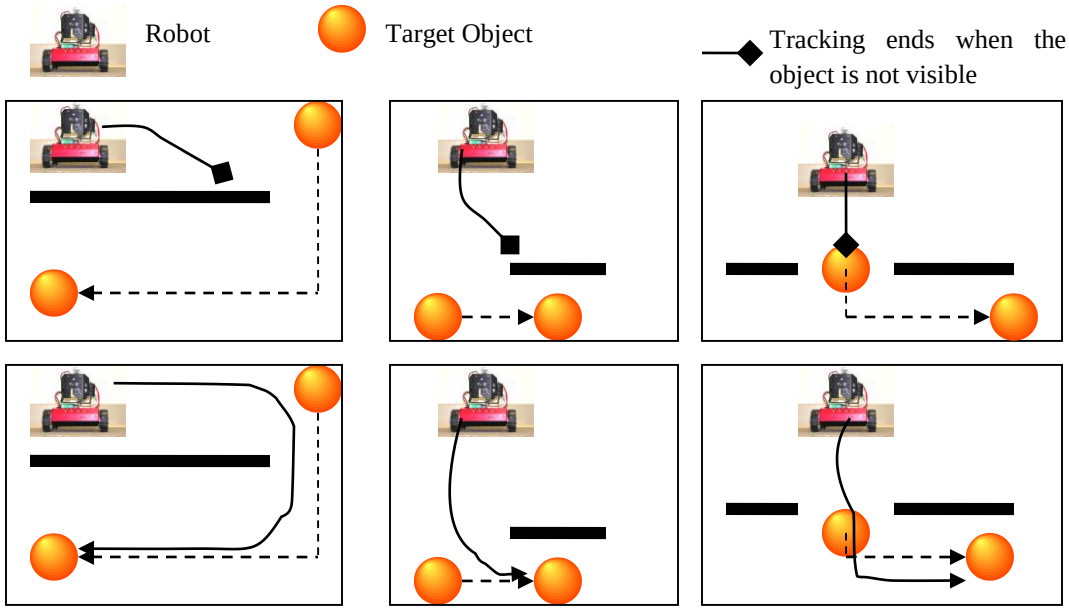


Fig. 2. Top row: 3 scenarios where object tracking fails. Bottom row: 3 scenarios how the tracking should be performed for the corresponding cases.

In our research, there are two phases of tracking: a) real object tracking (ROT) and b) virtual object tracking and exploration (Fig. 3). In ROT, the object is in the FOV when the tracking starts. However, the tracker may successfully reach the object or the object may be occluded before reaching the object. If ROT is not successful, the VOT phase starts. In VOT, the tracker tries to reach the latest visible location of the object. The last part of VOT is to determine the direction of the exploration. Then, the tracker starts searching for the object. When the object is detected, the ROT phase starts. The tracking continues until the object is reached.

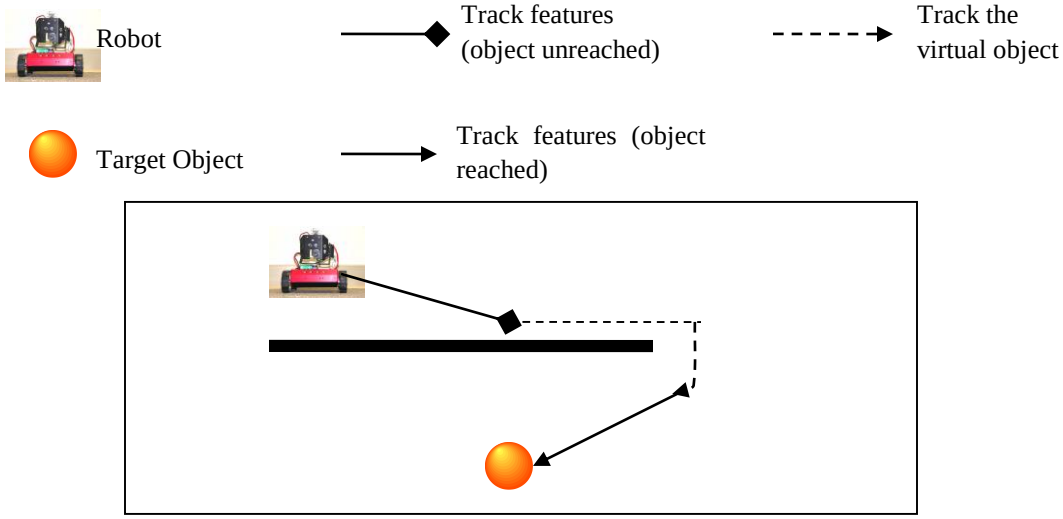


Fig. 3. Tracking phases.

3. VIRTUAL OBJECT TRACKING

The virtual object tracking (VOT) phase starts when real object tracking (ROT) phase (Fig. 4(a)) ends, Virtual object is the process of tracking the last appearance of the object (LAO). This phase includes the distance estimation to the LAO, reorientation, traveling the estimated distance, estimating the direction of the turn, and rotating towards that direction.

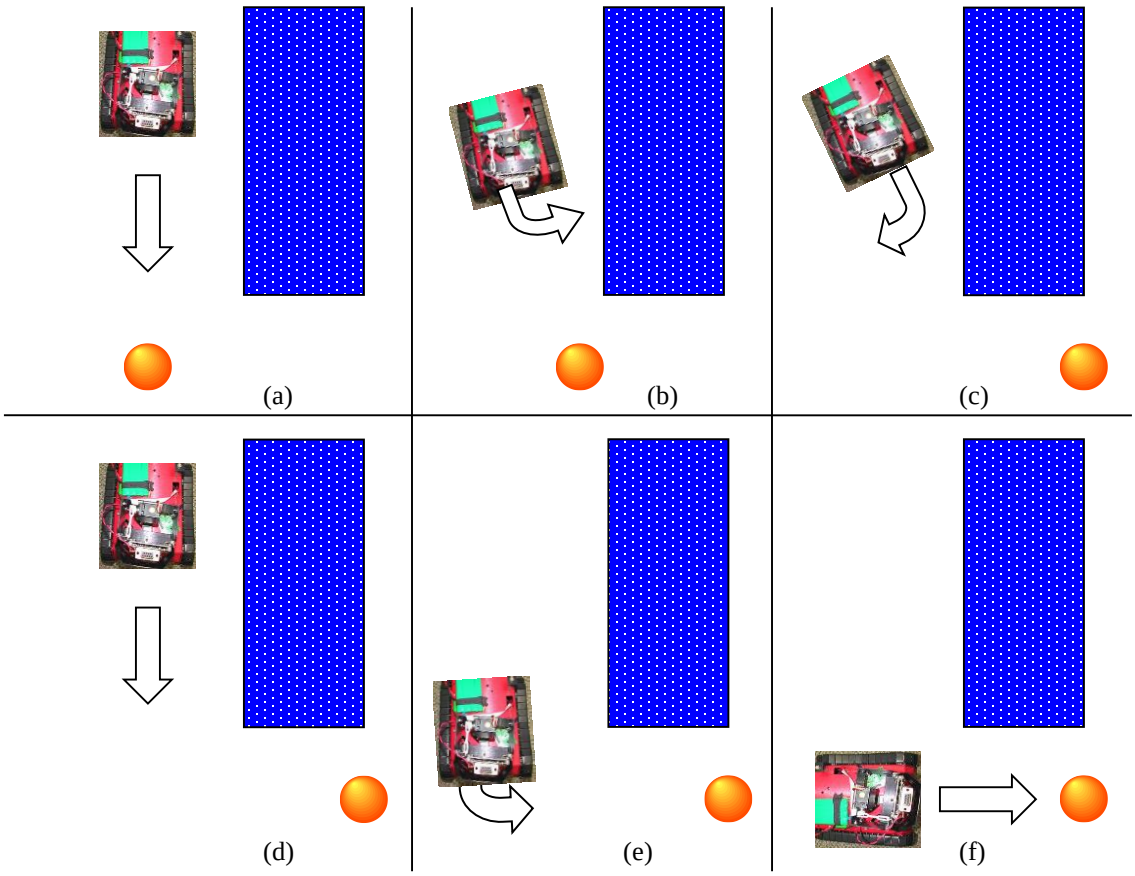


Fig. 4. (a) Robot starts tracking; and when the object is located straight ahead, the width of the object is constantly used to calculate the distance between the robot and the object, (b) robot tilts to left to centralize the object as it tracks the object that goes out of the field-of-view, (c) object is out of field-of-view; robot faces the wall; and re-orientation starts to avoid the wall, (d) distance is estimated to the virtual object and distance is traveled, (e) robot estimates the direction to turn and turns in that direction, and (f) object is visible again and regular tracking starts.

3.1. *Distance Estimation*

The only sensor that was available to the robot is the CmuCam2+.[11] There is no sonar or other means of directly determining the distance of an object. Thus, it is necessary to find some means of determining the distance to the object by interpreting the input from the camera. We have considered the width of the object as well as the area that is covered by the object.

Fig. 5 depicts the pinhole camera model where f represents the focal length, d represents the distance of the object, (a,b) represents the (x,y) coordinates of the object in the real world, and (a',b') represents the coordinates of the object in the image plane of the camera. CmuCam2+ is equipped with OmniVision OV6620 CMOS sensor, which can capture images of 352 by 288. The image plane of the sensor is 3.1 x 2.5mm. Each pixel size is 9.0 x 8.2 μm. The distance of the object can be estimated from the following equations:

$$\frac{a'}{b'} = \frac{a}{b} \tag{3.1}$$

or

$$\frac{a}{a'} = \frac{d}{f} \tag{3.2}$$

or

$$\frac{b}{b'} = \frac{d}{f} \tag{3.3}$$

Using Eq. (3.2) and Eq. (3.3), the distance can be estimated as

$$d = \frac{a}{a'} f \tag{3.4}$$

or

$$d = \frac{b}{b'} f \tag{3.5}$$

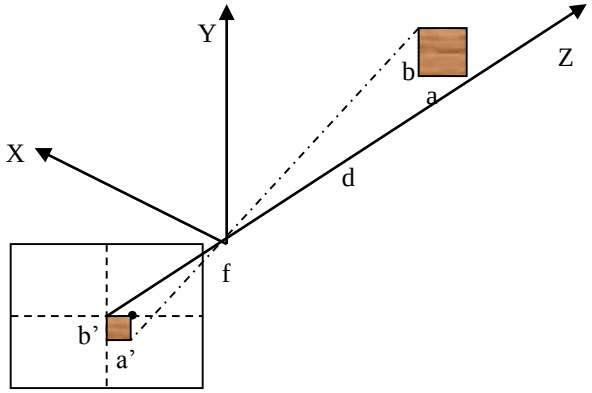


Fig. 5. Pinhole camera model

CmuCam2+ sensor also returns the number of pixels (area) that belong to an object. The number of pixels for a rectangular object (o) can be computed as a'x b' assuming that (a',b') corresponds to the top-left corner of the object whereas the bottom-left corner passes through Z. Using Eq. (3.4) and Eq. (3.5), we get

$$d^2 = \left(\frac{a}{a'} f\right) x \left(\frac{b}{b'} f\right) = \frac{(a \times b)}{(a' \times b')} f^2 = \frac{o}{o'} f^2 \tag{3.6}$$

$$d = \sqrt{\frac{(a \times b)}{(a' \times b')}} f = \sqrt{\frac{o}{o'}} f \tag{3.7}$$

We have considered the number of pixels for the object as well as the width of the object in our empirical studies. We have used an orange colored object having dimensions approximately 4" by 3". Table 1 provides the results for total number of pixels with respect to the distance. Table 2 presents the width of the object in number of pixels with respect to the distance.

Table 1. Results of test using total number of pixels

Distance (Inches)	11	16.5	22	27.5
Number of Pixels	191.4	109.2	73	42.6

Table 2. Results of Test Using Width in Pixels

Distance (Inches)	4	5	6	7	8	9	10	11	12
Number of Pixels	66	54	49	41	36	34	30	27	26

Our experiments reveal that the number of pixels is not a good measure for estimating the distance of the object due to the lighting and orientation of the object. The number of pixels is sensitive to the lighting as well as the orientation. Therefore, we decided to use the width to estimate the distance of the object.

According to our experimental results, the focal length *f* is around *0.9mm*. The distance is therefore can be computed as

$d = \frac{3}{a'} 0.9$ where a' is the width of the object in mm and distance d is in inches.

The distance is therefore approximately computed as $d = \frac{66}{a'} * 4$ where *a'* is the width in terms of pixels.

To estimate the distance of the object as reliably as possible, a history of distance measures is maintained in the memory. Whenever the count of distances exceeds the buffer size, the earliest one is removed from the history. Our algorithm for estimating the distance is provided in Algorithm 1.

Algorithm 1. The algorithm for estimating distance

```
estimateDistance(IN x2_g, IN x1_g, OUT actualDistance)
// x2_g: right x-ccordinate of the object
// x1_g: left x-ccordinate of the object
// width: x2_g-x1_g
begin
  width=x2_g-x1_g;
  if (width < 66 && width > 3) // check if width is reasonable
    if (distances buffer >= 30) // check if history buffer is full
      Remove the oldest distance from the history
    endif
    if (count of distances < buffer size)
      Add new distance as ((66 / width) * 4) to the history
    endif
    // find the average distance
    Sum of distances = 0;
    foreach (distance in history)
      Sum of distances += distance;
    end foreach
    actualDistance = sum of distances / size of history;
  endif
end
```

3.2. *Reorientation*

Generally, when the object disappears, it was moving either to the left or to the right. These movements to the left or right prompt the robot to rotate either left or right as well, in order to maintain a view of the object. This tracking may cause the robot to be directed toward a wall or another object at the time the object exits (Fig. 4(b)). Thus, simply proceeding forward the distance to the object and rotating is not sufficient to relocate the object. Some method must be used to reorient the robot so that it can proceed forward without running into the obstacle. The algorithm for reorientation is given in Algorithm 2.

Our method uses a timer to record the duration of any turn taken while tracking the object. If the object leaves the field of view during or shortly after a turn, the robot can then simply turn back in the opposite direction for an amount of time equal to that of the initial turn. In this way, it can be made certain that the robot is not left facing an obstacle as a result of previous tracking behavior. Thus, the robot will rotate back so that it is facing a direction where the object was previously located straight ahead (Fig. 4(c)).

Algorithm 2. Reorientation Algorithm

```
Reorient(robot)
// searching (A,B): returns true if A is searching for B at the moment
// rotating (A): returns true if A is true
// reoriented(A): returns true if A completed reorientation
// known(direction): returns true if direction is known
// ctd: boolean variable set to true if computing turning duration is
//      necessary
// time(X): returns when X happened
// exitDir: exit direction of the object
begin
  if (not searching(robot, object) and rotating(robot)
      and not reoriented(robot))
    // Initialize the stopping condition etc
    if (ctd)
      turnDuration = time(endedTurning)-time(startedTurning);
      stopTurningAt = time(Now) + turnDuration;
      ctd=false;
    endif
    // Reorient
    if (known(exitDir) and time(Now)<stopTurningAt)
      // NOTE: Reorientation is in the opposite direction of the exit
      if (object exited from right)
        TURN LEFT
      else if (object exited from left)
        TURN RIGHT
      endif
    else
      STOP REORIENTATION
      set turning as complete;
      set reoriented as true;
      set the last exit direction;
      reset the current exit direction;
    endif
  endif
end
```

3.3. *Traveling the Estimated Distance*

The robot has no functional method of traveling a specified distance. Thus, even after the distance to the object is determined, there must be a method devised by which the robot can travel the correct distance.

We decided to measure the speed of the robot under normal circumstances. Our goal is to determine the amount of time required to travel one inch. A simple timer was utilized to allow the robot to travel a specified amount of time. This timer provided accuracy to the millisecond. Trial and error tests were then performed to determine the appropriate amount of time for the robot to travel per second. Ultimately it was determined that the robot travels one inch in approximately 65 milliseconds. Basically, its velocity is 1/65=0.0154inches/ms. Thus, simply multiplying 65 by the number of inches to travel gives the appropriate amount of time to travel in milliseconds. Then the robot travels for this period (Fig. 4(d)). The algorithm is given in Algorithm 3.

Algorithm 3. Algorithm for traveling the distance

```
travelDistance(IN distance)
begin
  timeToTravel = distance * 65;
  stopAt = time(Now) + timeToTravel;
  if time(Now) < stopAt)
    TRAVEL FORWARD (until stopAt)
  endif
end
```

3.4. *Direction Determination*

Another required piece of information is the direction in which the object was traveling. Again, the only possible way to determine this information is via input from the CmuCam2+. Our method simply records at which edge of the field-of-view the object was present before it disappeared. That is, if the object was at the left edge of the frame and then disappeared, it must have exited to the left, and vice-versa. Algorithm 4 gives the pseudo-code for setting parameters and determining the exit direction.

3.5. *Rotation to the Direction*

Obviously, once the robot has traveled the appropriate distance and determined in which direction the object exited, it is necessary to rotate the appropriate amount in the appropriate direction. As with the problem with distance traveling, there is no feedback from the robot indicating what distance has been traveled. Thus, some means of rotating the correct distance must be devised. Similar to the solution to distance traveling, a timer was used to turn a specified amount of time. It was determined that 2500- millisecond (or 2.5 seconds) duration was appropriate for a rotation of about 90 degrees. Then, the robot rotates for the duration of the estimated time (Fig. 4(e)). Algorithm 5 gives the algorithm for setting parameters for rotation to the direction.

Finally, after rotating to the estimated direction, the robot searches for the features to track the object. When those features are detected, real object tracking phase starts (Fig. 4(f)).

Algorithm 4. Exit Direction Determination

determineDirection(centerx,centery): determines the exit direction based on the center of the object; then resets the parameters; and turns in that direction

```
// ctd: boolean variable set to true if computing turning duration is
//      necessary
// time(X): returns when X happened
// exitDir: exit direction of the object
// exitFromLeft: set true if object exits from the left
// exitFromRight: set true if object exits from the right

// (centerx, centery): (x,y) coordinates of the center of the object
begin
  if (centerx is on the left) // (centerx < 24)
    if (exitFromLeft)
      endedTurning = time(Now);
    else
      // the robot was not exiting left
      reset endedTurning to time(Now);
      reset startedTurning to time(Now);
      set last exitDirection to left;
      set ctd to true;
      set isTurning to true;
    endif
    TURN LEFT;
  endif
  if (centerx is on the right) // (centerx > 64)
    if (exitFromRight)
      endedTurning = time(Now);
    else
      reset endedTurning to time(Now);
      reset startedTurning to time(Now);
      set last exitDirection to right;
      set ctd to true;
      set isTurning to true;
      TURN RIGHT;
    endif
  endif
end
```

Algorithm 5. Algorithm for Setting Parameters for Rotation

```
setParametersForRotation()
begin
  set not reoriented;
  set duration computation as necessary;
  startedTurning = time(Now);
  endedTurning = time(Now)+ 2500ms;
  set isTurning to true;
  if (reorientiion to Left)
    exitDirection = Right;
  else
    exitDirection = Left;
  endif
end
```

4. MATHEMATICAL MODELING OF THE TRACKING

In this part, we give the mathematical modeling of the perfect tracking and our robot's tracking algorithm to compare the ideal path and the path of the robot. In order to provide a precise mathematical model, in this part we assume the robot and object moves on the Cartesian coordinate system (*xy-coordinate* system). In Section 4.1, we provide the modeling for real object tracking. In Section 4.2, we provide the modeling for virtual object tracking.

4.1. *Mathematical Modeling of Tracking When Object is Visible*

We provide the derivation of the ideal tracking as a system of ordinary differential equations. Instead of solving, we discretize it and obtain a system of difference equations which gives an approximate algorithm for the ideal tracking. Then, we give our algorithm via difference equations and we revise it to include some application errors due to some external factors. We assume that the object is always in robot's sight and the hallway is sufficiently wide for robot's motion.

4.1.1. *Mathematical Modeling of the Ideal Tracking*

Let $r(t) = (x(t), y(t))$ be the location of the robot and $o(t) = (a(t), b(t))$ be the location of the object at time t. So they will make a curve in xy system with respect to time t. We assume that the robot has a constant speed regardless of the target's speed, i.e.,

$$\| r'(t) \| = \sqrt{x'(t)^2 + y'(t)^2} = v \tag{4.1}$$

for some constant speed v.

If the robot follows the object in a perfect manner, the direction of the robot at any time should be exactly through the object. This tells us that the tangent vector of *r(t)* at anytime *t* is parallel to the vector *o(t)* - *r(t)*. So their unit vectors should be same:

$$\frac{r'(t)}{\|r'(t)\|} = \frac{-r(t) - o(t)}{\|r(t) - o(t)\|} \tag{4.2}$$

Since $\| r'(t) \| = v$ (constant speed anytime), we get the following system of ordinary differential equations (ODE) from Eq. (4.2):

$$\frac{x'(t)}{v} = -\frac{x(t) - a(t)}{\sqrt{(x(t) - a(t))^2 + (y(t) - b(t))^2}}$$

$$\frac{y'(t)}{v} = -\frac{y(t) - b(t)}{\sqrt{(x(t) - a(t))^2 + (y(t) - b(t))^2}} \tag{4.3}$$

The exact solution of this ODE system might not be obtained easily. For this reason, it is practical to discretize it and get the approximate values of *(x(t), y(t))* at discrete points. Discretizing the system given in Eq. (4.3), we get Eq. (4.4):

$$\frac{x_{n+1} - x_n}{t_{n+1} - t_n} = -v \frac{x_n - a_n}{\sqrt{(x_n - a_n)^2 + (y_n - b_n)^2}}$$

$$\frac{y_{n+1} - y_n}{t_{n+1} - t_n} = -v \frac{y_n - b_n}{\sqrt{(x_n - a_n)^2 + (y_n - b_n)^2}} \tag{4.4}$$

where $x_n = x(t_n)$, $y_n = y(t_n)$, $a_n = a(t_n)$ and $b(t_n) = b_n$.

Given the initial locations of the robot and the object, one can find almost exact path of the robot in the ideal tracking, using the algorithm given above if t_{n+1}-t_n is taken very small. However, this may not be practical.

4.1.2. *Mathematical Modeling of Robot's Tracking*

Eq. (4.5) provides the mathematical modeling of our tracking algorithm. At each step the robot checks if the location of the object is on the right or left of its direction. The robot makes a turn of angle α to the right or left or does not change its direction. After changing its direction it moves straight until it checks the object again. In below k_n (Eq. (4.6)) denotes whether the robot's next move will be to the right or left:

$$\frac{x_{n+1} - x_n}{t_{n+1} - t_n} = v \frac{(x_n - x_{n-1})\cos(k_n \alpha) - (y_n - y_{n-1})\sin(k_n \alpha)}{\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}}$$

$$\frac{y_{n+1} - y_n}{t_{n+1} - t_n} = v \frac{(x_n - x_{n-1})\sin(k_n \alpha) + (y_n - y_{n-1})\cos(k_n \alpha)}{\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}} \tag{4.5}$$

where

$$k_n = \begin{cases} 1 & \text{if } b_n - y_n > \dfrac{y_n - y_{n-1}}{x_n - x_{n-1}}(a_n - x_n) \\ 0 & \text{if } b_n - y_n = \dfrac{y_n - y_{n-1}}{x_n - x_{n-1}}(a_n - x_n) \\ -1 & \text{if } b_n - y_n < \dfrac{y_n - y_{n-1}}{x_n - x_{n-1}}(a_n - x_n) \end{cases} \tag{4.6}$$

We also note that the model above assumes that the robot sticks to our algorithm in a perfect manner. However, this is not generally the case in practice. In order to take into account the possible errors in motion, we can revise this model by adding possible errors. There are two types of errors that should be considered: rotation angle error (α'_n) to centralize the object and speed error (v'_n). The errors in motion can result at turns and straight moves. When making a turn to the left or right the robot might not be precise at making α turn. So it brings an error term to α. At each step we can add a random error α'_n to α. Another ambiguity is that when robot makes a straight move, it might not go with a constant speed. In this case, in the model we add a random error v'_n to the speed v at every step. So the model can be revised as Eq. (4.7):

$$\frac{x_{n+1} - x_n}{t_{n+1} - t_n} = (v + v'_n)\frac{(x_n - x_{n-1})\cos(k_n(\alpha + \alpha'_n)) - (y_n - y_{n-1})\sin(k_n(\alpha + \alpha'_n))}{\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}}$$

$$\frac{y_{n+1} - y_n}{t_{n+1} - t_n} = (v + v'_n)\frac{(x_n - x_{n-1})\sin(k_n(\alpha + \alpha'_n)) + (y_n - y_{n-1})\cos(k_n(\alpha + \alpha'_n))}{\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}} \tag{4.7}$$

With this error included in the model, the motion of the robot can be simulated. Furthermore, it is possible to determine how much the robot's motion differs from the ideal tracking via simulating both mathematical models.

4.2. *Modeling of Virtual Object Tracking*

In this section, we deal with the case in which the robot does not see the object at a step. We provide the analysis for the robot following the object in an L-shaped corridor. Donald[21] also provides an analysis with respect to the geometric constraints of the environment. The object becomes invisible when the object turns around the corner. We deal with the problem of finding a safe turning angle and a safe region (around the corner) that enables the robot to see the object again.

In this case the robot has to make a sharp turn to capture the view of the object. In our experiments we roughly know the path of the object. The object moves on an L-shape corridor. So the object might disappear from the sight of the robot when it turns left (or right) at the corner. In this case the robot moves toward the object's last location before it got out of the robot's sight. Now let (a_n, b_n) be the point where the object stands when the robot stops to turn the corner to see the object (call this point (x_n, y_n)). When the robot reaches (x_n, y_n), the robot should make a left turn to see the object. Now we want to find the minimum angle which makes sure that the robot sees the object. Let θ be the angle of range of sight. Since its previous location is (x_{n-1}, y_{n-1}), the robots direction will be in the direction of $(x_n - x_{n-1}, y_n - y_{n-1})$. Let us call minimum turning angle of the robot r which is required to see the object at the point (a_n, b_n). Using the rotation and line equations, we get Eq. (4.8):

$$\frac{b_n - y_n}{a_n - x_n} = \frac{(y_n - y_{n-1})\cos(r + \theta)}{x_n - x_{n-1} - (y_n - y_{n-1})\sin(r + \theta)} \tag{4.8}$$

By using Eq. (4.8), the required minimum turning angle for the robot can be computed with respect to the location of the object. If a possible region for (a_n, b_n) is given then by using the above equality a feasible set of the values of r can be given.

Due to the robot's imperfect nature, the location of the robot at the corner might not be precise. There are four types of uncertainties involved: a) computing the distance to travel, b) the actual distance that is traveled, c) reorientation angle, and d) rotation angle. Computing distance to travel and traveling this distance yields an error on distance. If the desired distance is represented with c, the range of distance can be assumed to be within $(c \pm d)$ where d is an error with respect to c. To secure the rotation angle r, one should also determine the region where the location of the robot (x_n, y_n) lies. Let the error $-\beta < \alpha'_n < \beta$ and $-d < c'_{n-1} < d$. So the region that contains the point (x_n, y_n) formed within these error bounds makes a part of an annulus as seen in Fig. 6. The location of the (x_{n-1}, y_{n-1}) can be between the right wall and the left wall. We take the worst case which is (x_{n-1}, y_{n-1}) is next to the left or right wall. In this case the width of the horizontal leg of the corridor should be at least $c + d - (c - d) \cos 2\beta$ and the width of the vertical leg of the corridor should be at least $(c + d) \sin 2\beta$ where c is the expected distance between (x_n, y_n) and (x_{n-1}, y_{n-1}).

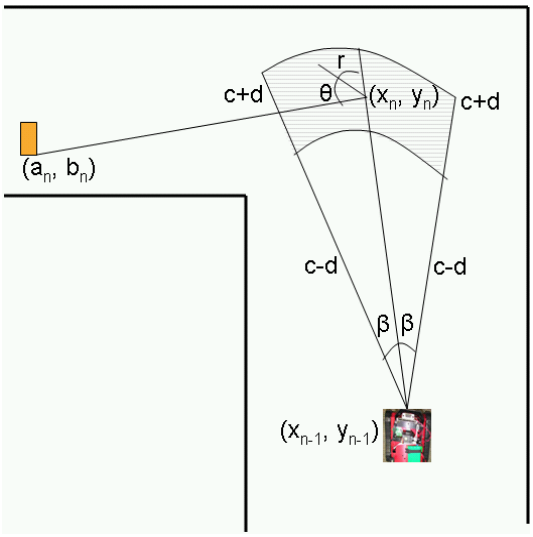


Fig. 6. Geometric modeling of the environment.

5. EXPERIMENTS

5.1. *System Components*

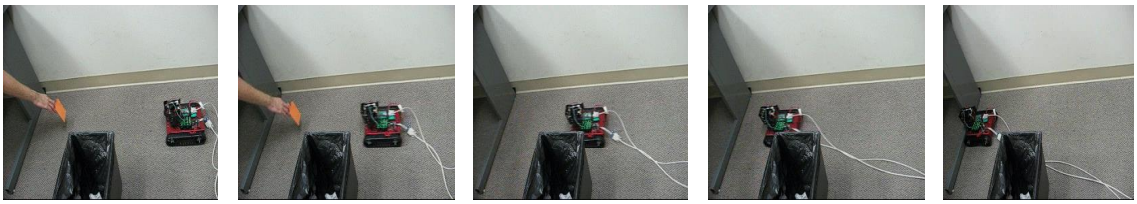
We have built a Traxter-type robot[12] to track color objects. We have used Serializer .NET controller[13] to control the robot and CmuCam2+ camera[11] to track objects. The robot was able to move using two motors and two sets of tracks, and the robot was able to "see" by using the CmuCam2+ camera fitted atop the Traxter. The CmuCam2+ camera is capable of visually tracking objects based on the color of the object. No other sensors or motors were fitted to the robot. The software to control the robot was written in C# using the .NET Framework. The software developed was executed on a Windows XP-based PC fitted with two RS232 ports. Two RS232 ports were needed, as both the Traxter controller and the camera each require a serial connection for control.

5.2. *Experiments*

Our robot does not have any specific assumptions on the speed of the object. However, we assume that the object is not too fast where it is almost impossible to capture it. We use the color feature to track the object. The color is used in general sense here. Gray-scale objects can also be tracked by the robot. The limitation is that the object should have a distinguishable color than the environment. If the object has multiple colors,

one of the colors may be chosen to track continuously. Or the problem can be further resolved by performing further image processing. A template of the object can be used to track the object.

We have tried several ways of tracking an object that leaves the FOV. Initial idea was to continue in the direction of the last appearance of the object. However, this was not close to a solution. A sample sequence is provided in Fig. 7.



(a) Object is visible. (b) Tracking starts. (c) Object is not visible. Continues moving. (d) Object is not visible. Continues moving. (e) Hits an obstacle.

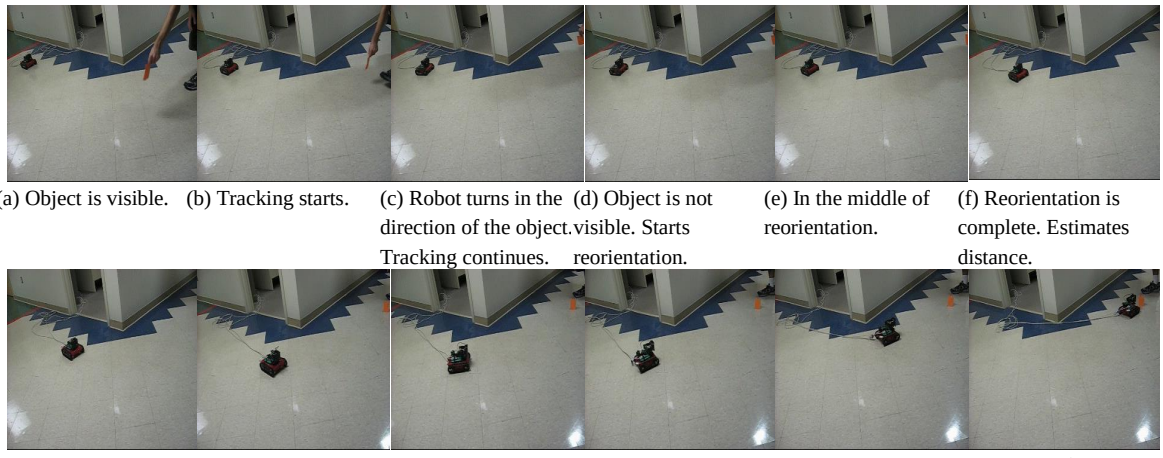
Fig. 7. Tracking continuously in the direction of the object.

Later we added several steps to our initial algorithm including distance estimation and traveling the required distance. In that case (without reorientation), our robot hit the walls. This made clear that reorientation was a necessary step for virtual object tracking. Fig. 8 displays the necessity of reorientation.

(a) Object is visible. (b) Tracking starts. (c) Robot turns in the direction of the object. Tracking continues. (d) Object is not in the FOV. (e) Tracking continues based on last orientation. (f) Hits the obstacle.

Fig. 8. The necessity of reorientation.

Finally, we added reorientation to our methodology and performed experiments. Fig. 8 shows a sample sequence of one of our experiments with successful tracking. Fig. 9(d)-(f) shows the reorientation.



(a) Object is visible. (b) Tracking starts. (c) Robot turns in the direction of the object. Tracking continues. (d) Object is not visible. Starts reorientation. (e) In the middle of reorientation. (f) Reorientation is complete. Estimates distance.

(g) Travels estimated distance. (h) Traveling continues. (i) Turns to the estimated direction. (j) Detects the object. (k) Tracking starts. (l) Object is found.

Fig. 9. Our method with reorientation and virtual object tracking.

The videos of our experiments are available at http://www.cs.uah.edu/~raygun/objecttracking.htm.

5.3. *Discussion*

Our current system assumes that the object can be distinguishable from the background. This also indicates there should not be a) a reflection on the object on the surfaces and b) another object with similar properties of the target object. In addition, our system currently determines the distance based on the size of the object. The size of the object is determined by the color of the object. The color depends on the lighting of the environment. If the lighting changes, this may lead to incorrect distance estimations. The distance estimations can be performed using other types of sensors. Our algorithm is not designed for objects that change their shape or size. In those cases, our distance estimation to the object would be affected. A proximity sensor should be used to track objects that change their shapes or sizes. However, our algorithm can still be used when such a sensor is not available or it fails. In our system, collision is considered as failure. An ideal system should be able to handle collisions and obstacles. A proximity sensor would be very helpful to avoid collisions.

It is reasonable to use the latest and all possible technology available for tracking. For example, GPS or sonar sensors could be used. Using GPS may resolve some of the problems such as position determination or route following. However, we should also consider the cases where GPS signals are not received due to bad weather conditions or basically the signals could be jammed. In those cases, relying on visual information could be the only option.

Ideally, we expect the robot to follow the exact path the object goes. If the object avoids obstacles, our robot should also avoid obstacles. Otherwise, a different model needs to be developed for every different environment. This raises the question: Should the robot be aware of the environment or the object? We propose that the robot should focus on the object. If the object can follow a path, our robot should also be able to follow that path. The current limitation is that after robot rotates towards the direction that object disappeared, the object should be visible when the robot completes the rotation.

In this paper, we divided object tracking into two phases: real object tracking and virtual object tracking. Real object tracking deals with tracking using detectable features of the object. Different applications have various challenges and techniques to deal with this first phase. For automated surveillance systems, tracking people is an important application. These applications may require tracking people in a crowd. For these applications, object tracking may use point tracking, kernel tracking, or silhouette tracking algorithms using features like color, edges, optical flow, and texture[1].

The second phase is virtual object tracking. This phase considers tracking based on last visible or detectable position of the object. In this paper, we propose to reach the last detectable position of the object. For example, consider an unmanned aerial vehicle (UAV) that tracks an object such as a car. If the car disappears at the entrance of a tunnel, the UAV should first reach the point where the car disappeared. This includes the computation of distance to the last visible point and then traveling to that location. We should note that each application has its own requirements. UAV has more freedom of movement than a car. A UAV may get the benefit of this freedom. For an application where there is a leader car (or object) and a follower car (or object), the follower has almost the same freedom of movement as the leader. It may enter the tunnel as the leader car. However, another car may get in between follower and the leader car. In such a case, a strategy needs to be developed to continue motion. UAV is not affected by such a case (i.e., a car gets in between). However, it must resolve VOT when the car enters a tunnel. For surveillance systems, similar cases may also occur.

The virtual object tracking phase finishes with the exploration stage. During exploration, the tracking system should determine what to do to get into real object tracking phase again. Basically, it must search for the object. For UAV example, probably it would be a good idea for the UAV to go to the other end of the tunnel. For a following car, it should pass the car in between.

These phases may have differences for various applications. Our method states that there are important steps involved in these phases. The problem of VOT and exploration should be studied carefully. In this paper, we point out important steps involved for VOT such as distance estimation, reorientation, traveling the distance, direction determination, and rotation. There could be some differences how these steps are

implemented in different applications and environments. Data mining framework[23] can also be applied for determining the reorientation, direction, and rotation as future work.

Our methodology can be adapted to different applications with some modifications and improvements. A close application to ours is leader-follower problem where a follower object is required to track a leader object. This is usually studied for autonomous robot systems where a leader is followed by a single or a group of follower robots. Another example is autonomous vehicle navigation system where an autonomous vehicle may need to follow another vehicle on real road conditions. The proposed mobile system can also supplement passive multi-camera surveillance systems. Currently, most multi-camera surveillance systems are not mobile. They can only work as long as the object is in the field-of-view of one of the cameras. When the object leaves the field-of-view of all cameras, the mobile surveillance may come into the picture.

6. CONCLUSION

In this paper, we examined the problem of tracking objects that leave the field of view. The important phase in our methodology is the virtual object tracking. We have presented the steps to be included for virtual object tracking. One problem that we did not address in the paper was 'what if the object is still not visible after virtual object tracking ends'. In that case, a sophisticated exploration phase needs to be added. We plan to explore ways of tracking the object even if the object is not visible by estimating possible paths of the object. We have obtained quite interesting and promising results in our experiments. In addition, an advanced robot with various sensors to better estimate the distance of the object. Since our goal was to test our approach, the robot we built had satisfactory results to test our algorithm.

References

1. A. Yilmaz, O. Javed, and M. Shah, Object tracking: A survey, *ACM Comput. Surv.* **38**(4) (2006).
2. J Fritsch, M Kleinehagenbrock, S Lang, GA Fink, and G Sagerer, Audiovisual person tracking with a mobile robot, *Proc. Int. Conf. on Intelligent Autonomous Systems*, 2004, 898-906.
3. C.-H. Lin, C.-H. Yang, C.-K. Wang, K.-T. Song, and J.-S. Hu, A new design on multi-modal robotic focus attention, *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, 2008, 598-603.
4. H. Zender, P. Jensfelt, and G.-J.M., Human- and Situation-Aware People Following, *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on* , 2007, 1131-1136.
5. M. Liem, A. Visser, and F. Groen, A hybrid algorithm for tracking and following people using a robotic dog. In *Proceedings of the 3rd ACM/IEEE international Conference on Human Robot interaction* Amsterdam, 2008, (Amsterdam, The Netherlands, 2008) 185-192.
6. S. Ekvall, D. Kragic, and F. Hoffmann, Object recognition and pose estimation using color cooccurrence histograms and geometric modeling, *Image Vision Comput.* **23**(11) (2005) 943-955.
7. F. Belkhouche, and B. Belkhouche, On the tracking and interception of a moving object by a wheeled mobile robot, *Robotics, Automation and Mechatronics, 2004 IEEE Conference on* , Vol. 1, 2004, 130-135.
8. B. Tova, S.M. La Valle, and R. Murrieta, Optimal navigation and object finding without geometric maps or localization, *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, Vol. 1, 2003, 464-470.
9. Y. Gabriely, and E. Rimon, C-space characterization of contact preserving paths with application to tactile-sensor based mobile robot navigation, *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* , 2008, 1792-1797.
10. C.-H. Chen, C. Cheng, D. Page, A. Koschan, and M. Abidi. Tracking a moving object with real-time obstacle avoidance. *Industrial Robot: An International Journal* **33**(6) (2006) 460-468.
11. CmuCam2+ vision sensor (2011), http://www.cs.cmu.edu/~cmucam/.
12. Robotics Connection, (2009), http://www.roboticsconnection.com/p-15-traxster-robot-kit.aspx.
13. Serializer .NET Controller (2009), http://www.roboticsconnection.com/p-16-serializer-robot-controller.aspx.
14. Y. Huang, and M. Essa, Tracking multiple objects through occlusions, *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* , Vol. 2, (2005, 1182.
15. B. Leibe, K. Schindler, N. Cornelis, and L. Van Gool, Coupled Object Detection and Tracking from Static Cameras and Moving Vehicles, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , **30**(10) (2008) 1683-1698.
16. O. Lanz, Approximate Bayesian Multibody Tracking, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **28**(9), (2006) 1436-1449.

17. S. Pellegrini, A. Ess, K. Schindler, and L. van Gool, You'll never walk alone: Modeling social behavior for multi-target tracking, *Computer Vision, 2009 IEEE 12th International Conference on* , 2009, 261-268.
18. A. Ess, B. Leibe, K. Schindler, and L. van Gool, Robust Multiperson Tracking from a Mobile Platform, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , **31**(10) (2009) 1831-1846.
19. B. Jung, and G.S. Sukhatme, Real-time Motion Tracking from a Mobile Robot, *International Journal of Social Robotics* , **2**(1) (2010) 63-78, DOI: 10.1007/s12369-009-0038-y.
20. H. Tao, H.S. Sawhney, and R. Kumar, Object tracking with Bayesian estimation of dynamic layer representations, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , **24**(1) (2002) 75-89.
21. B.R. Donald, A Geometric Approach to Error Detection and Recovery for Robot Motion Planning with Uncertainty, *Artificial Intelligence*, **37**(1-3) (1998) 223-271.
22. S. Li and F. Ren, Realizing Face-to-Face Interaction by View-Based Tracking, *International Journal of Information Technology and Decision Making*, **1**(2) (2002) 331-347.
23. Peng, Y., Kou, G., Shi, Y., and Chen, Z., A Descriptive Framework for the Field of Data Mining and Knowledge Discovery, *International Journal of Information Technology & Decision Making*, Vol. 7, Issue: 4, Page 639 - 682, 2008