Technical University of Delft

Bachelors Project

Efficient Person Tracking based on Depth data

Author:
Hans Gaiser

Supervisor:
Dr.ir. D.J. Broekens
Dipl.ing. M. Rudinac

Bachelor thesis submitted in partial fulfillment of the requirements of the Bachelor of Science at the Technical University of Delft, department of Computer Science.

July 23, 2012

Abstract

This paper introduces an alternative approach to following persons for robotic systems in a domestic environment; Flood Fill on Depth (FFOD) data. Other related work is shown: tracking using a Lucas Kanade tracker, tracking using OpenTLD (**T**racking, **L**earning, **D**etecting) and tracking using a state of the art particle filter. These algorithms are all processing RGB images, while this paper suggests an alternative: image processing on depth data. The FFOD algorithm is evaluated by checking the introduced requirements and benchmarking using the Bonn Benchmark on Tracking (BoBoT). FFOD passed all critical requirements but lacks a detection system to regain a person when lost. The BoBoT benchmark resulted in a precision of 80.09%, which is almost 10% higher than a state of the art particle filter algorithm. Hereby proving that person following using depth data is an interesting approach and should be further researched.

1 Introduction

There have been many efforts, and progress, in the field of robotics over the last years in an attempt to take the robot out of its industrial cages and into the comfort of the household. This is not an easy task, as it turned out, and it has been only recently that the first type of robots can be found in some homes. One of them being the robotic vacuum cleaner: Roomba. The Roomba is capable of moving around a room and clean dirt as it goes along. The actual robotic part about the Roomba is that it uses infra-red sensors to determine its location and sensors to detect dirt that needs cleaning[10]. Navigating through an environment is only one of the tasks robots have to do in order to be useful around the house. Service robots are generally designed to assist people in tasks within the household and follow commands from their owners. These commands could contain the owner showing the robot what and where it has to do a certain task. For instance in the case of the Roomba, it could be desirable to tell it to clean a certain area again. This means the robot has to know how to follow the person. Following persons is not an ambiguous task, there might be more than one person walking around the robot, the environment can be very cluttered with other objects, parts or the whole of a person can be visually obstructed[22].

This paper will discuss various methods for tracking a person around a cluttered environment. The problem will first be described in detail, after which the different existing methods will be introduced. Finally, an alternative will be presented: a simple but effective way of tracking persons using just depth data from the environment. This method will then be benchmarked using the BoBoT (Bonn Benchmark on Tracking)[4] data set and evaluated using additional image sequences. The BoBoT data also contains the ground truth, a manually entered bounding box which can be considered as the best fitting bounding box around the person. Person tracking algorithms can be benchmarked by calculating the overlap of the resulting bounding box with this ground truth bounding box. The average overlap is considered the precision of a tracker, which is what is used for comparison.

2 Problem statement

As described in the introduction, a robot needs to be able to follow a person through a cluttered environment. This environment is almost completely unknown. It is assumed for the testing purposes of the methods described later, that this environment would reflect a typical household. This includes furniture, people walking around and different lighting throughout the room. It is worth noting that obstacles that

are not high enough to visually obstruct the tracked person (such as coffee tables) do not affect the tracking, only the obstacle avoidance for the robot, but this is not a topic discussed in this paper.

This paper will suggest a simple alternative to existing methods, using Microsoft's Kinect device or other similar hardware to retrieve depth information about the surroundings. These devices are often very cheap, while still providing enough depth data. There has not been much research on this type of image processing in combination with person tracking yet, however as will be shown later on, it provides promising results. The definition of the problem this paper aims to resolve is the following :

Construct a robust and efficient way of tracking persons through a domestic environment.

This chapter will discuss measurable requirements for this problem statement. The requirements will be divided using the MoSCoW model[20]. The MoSCoW model divides the requirements in: must have, should have, could have, and won't have.

2.1 Must have

Must have requirements are requirements that the tracker must atleast fulfil before a method is considered successful. In this specific case the requirements are designed so that if all are fulfilled, the robot can do the most simplest of person tracking for a short amount of time (for this project a length of 10 seconds has been chosen).

Given an initial point or bounding box of a person, the tracker must ...

- ... successfully find a bounding box of this person in the same frame.
- ... not drift off target and keep track of the person for at least 10 seconds, while the scene changes are ignored. During this time the person is standing still.
- ... keep track of the person for 10 seconds while the person is moving forward, backward or sideways in front of the robot in an area a maximum of 5 meters away from the robot. During this time the robot will stand still and the person is not visually obstructed and facing the same direction as with initialization.
- ... keep track of the person for 10 seconds while the person is moving forward at a normal speed for robots, roughly 1m/s, and the robot is following at the same speed (the relative speed between the robot and the person is in this case 0m/s).
- ... keep track of the person while the person is walking away and within 5 meters of the front of the robot. During this time the robot is standing still.
- ... keep a frame rate of at least 20 frames per second while tracking on modern day systems[1], while frames are generated at 25-30 frames per second.

[1]For the benchmark and additional tests later on, an Intel system with dual core running at 2.0Ghz is used.

2.2 Should have

Should have requirements are requirements with high priority for the algorithm. These are generally very important requirements, but not critical ones. The algorithm can still function effectively if not all or none of these requirements are fulfilled.

- When both the robot and the tracked person are standing still and someone with a distinctively different colour clothing is walking in between, the tracker should either momentarily pause or keep the bounding box on the correct target and resume normally when the target is visible again.
- The tracker must keep track of the target for 10 seconds with any frame rate between 10-30 frames per second, while both are moving forward at the same speed.
- When the person turns around in place, the tracker should not lose the person.
- When the person turns around while moving, the tracker should not lose the person.
- If the tracked person is for any reason temporarily lost and reappears in the screen, the tracker should regain its target.

2.3 Could have

Could have requirements are considered desirable, but not necessary for the effectiveness of the algorithm. If enough time and resources are available, it is generally worth while to have a look at these requirements.

- If the person moves around a corner and is visually obstructed by the wall, the tracker should regain the person after it has cleared the corner and sees the person again.
- If the person changes appearance during tracking, by for instance putting on a coat, the tracker should not lose the person.
- If the person and the robot are standing still and someone who looks almost identical to the tracked person walks in between, the robot should afterwards still be tracking the original person.
- When the person is temporarily completely occluded, the tracker should estimate where the person will be.

2.4 Won't have

Won't have requirements are generally not implemented in the current release. They are requirements to which can be looked at in the future.

- If the person completely disappears in a crowd of people, the tracker will pause until this person reappears from the crowd.
- If multiple persons are walking in the view of the robot and they all look more or less identical, the robot should still keep track of the target person, regardless of the motion of the individuals.

3 Related work

This section will discuss some algorithms which are already used to follow persons. Following persons is not only used in the field of robotics, surveillance is also a popular application of this type of image processing[15]. A big advantage in this type of application is that the camera is generally standing still or hanging on a ceiling. This means the viewing angle and background are more or less static, which allows some lesser robustness in the initial detection and in the tracking.

Using vision to track persons has proven to be a difficult task since there can be many variations in illumination, angle and individual appearances[13]. There have been many attempts to track persons using vision[12, 8, 6] and also some using laser scanners[13, 16]. Laser scanners are less affected by the before mentioned problems that can occur, however they bring with them other difficulties. First and foremost, is that they are very expensive[14]. That is also the main reason they will not be discussed beyond this chapter in this paper, as they were not available for testing. Additionally, the current laser scanner algorithms[13, 16] depend on knowing how the environment looks like, in order to distinguish people from background, but this makes these algorithms less dynamic.

A more classical approach to object tracking is done using RGB images. Because RGB image processing has been around for a long time, there is better research, documentation and algorithms available. This paper will discuss three such algorithms. The first is the rather old, but successful, Lucas Kanade tracking. The second is the very new OpenTLD algorithm and the last is a Particle Filtering tracker.

3.1 Lucas Kanade

One of the more common means of tracking any object in RGB images is using the Lucas Kanade tracking algorithm. This algorithm has proven to be really effective over the years[1] and was first introduced in 1981. The Lucas Kanade tracking works by finding the optical flow between two sequential frames. It tries to deform one image so that it best matches the second image, thereby finding the transformation between the two. Lucas Kanade can therefore transform a set of points from one frame to the other. After initialization, a set of easily traceable points needs to be found which will be tracked to the next frames. An example of this process can be seen in Figure 1.

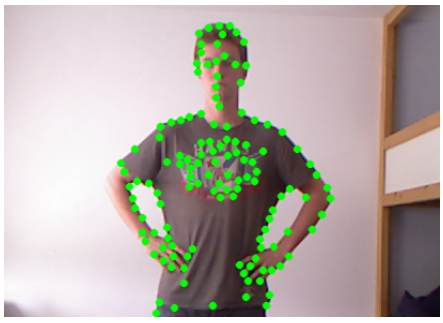

Figure 1: An example of the Lucas Kanade tracker. The green dots represent good features which the tracker can more easily track.

The OpenCV[21] library supports many algorithms, among them the Lucas Kanade optical flow tracker. It also contains an algorithm for finding feature points in an image. However it does not prune points that are no longer valid, for example

by drifting away. The algorithm that was constructed for testing uses the implementation of OpenCV and prunes points that are too far away from the majority of the points. This pruning is based on the ROS package called $pi\_face\_tracker$[7], which is used to track faces. It first calculates the mean squared error of all points to the center of gravity. Then for each point it calculates the ratio of its squared error with the center of gravity of all points. If this ratio is above a certain threshold, then the point is considered too far away and gets discarded. For the tests, this ratio-threshold has been set to 3.0. A higher threshold would result in too few points being pruned, too low would result in too many points being pruned.

A remark that has to be made for this algorithm is that it is purely tracking given points. If the points have been lost, through occlusion or fast movement for example, it will not try to regain these points. There should be another part to feed the tracker with new points to track when they become available.

The tracker is also easily influenced by changes in illumination, in very light or very dark scenarios it has difficulty finding good points to track or has difficulty tracking the already found points. Another flaw is that when the target person is wearing uniform coloured clothing, the tracking tends to let the few points that it did find, drift away to the edges of the target where they are more easily lost, making this tracker not very effective on uniform coloured clothing.

3.2 OpenTLD

OpenTLD (**T**racking, **L**earning, **D**etecting) is a very new algorithm, or actually a combination of algorithms, used to keep track of an object. The idea behind the algorithm is that the longer you let it run, the better it becomes at detecting the targeted object. It does this by constantly adjusting the detection classifier with positive patches from the target and negative patches from the background[11].

When the algorithm is initialized, it initializes the detector and the tracker simultaneously. Until the tracker loses the target or until it is no longer confident it has the correct position of the target, the tracker will 'tell' the detector what the correct position of the target is. The detector will also try to detect the target in the same frame, if it fails or if it is not similar to what the tracker gave as target location, the detector gets updated. It will keep a set of positive patches, patches from the target, and a set of negative patches, patches from the background that might look like the target, but are in fact something else. Every iteration these two sets get updated so that the detector is always up to speed on the current appearance of the target. When the tracker loses the target object, the detector will take over to detect the object in the screen. If it does eventually find the object, it will reinitialize the tracker and the cycle begins again.

When initially testing this algorithm it seemed to work good, however on further inspection some flaws were discovered. Both the detector and the tracker rely on feature points, similar to the Lucas Kanade tracker. Therein lies its weakness, because if someone has uniform coloured clothing, there are little or no detected features. Therefore in this case the tracker easily fails and the detector does not see enough features to regain the target again. At this point the algorithm will simply stop detecting or tracking anything any more. Having to track persons with uniform coloured clothing is not something that should be ignored. Additionally, the detector had big difficulties with malleable objects, such as a persons body while moving. It is not updating its classifier quick enough to keep up with the motion of the person, therefore if the tracker loses the target there is a high chance the detector will not be able to regain the target. It was found that this algorithm works fine on solid, smaller objects, but not on bigger malleable objects like a person.

3.3 Particle Filtering

The particle filter tracking method is somewhat newer than the Lucas Kanade tracking. It is based on the condensation (**con**ditional **dens**ity propag**ation**) algorithm, which was first introduced in 1998. The condensation algorithm is a probabilistic algorithm[9], which was designed to robustly track agile moving objects in a cluttered environment. A particle filter tracker exists of particles that are acting as a sort of probe on the image to detect the object. These particles can be modelled as a vector[12]:

$$\mathbf{x} = (x, y, w, h, v_x, v_y, C)^T$$

For which $\mathbf{x}$ is the particle, (x, y, w, h) the bounding box describing the tracked object, v_x and v_y its speed in x and y respectively and lastly C its classifier. Each particle is moving randomly on the image and their size changes randomly as well, both based on the so called motion model, using a Gaussian distribution. The motion model used in this algorithm is given as follows[12]:

$$\begin{aligned} v_{x,t} &= v_{x,t-1} + G(0, \sigma_x^2), \\ v_{y,t} &= v_{y,t-1} + G(0, \sigma_y^2), \\ x_t &= x_{t-1} + v_{x,t}, \\ y_t &= y_{t-1} + v_{y,t}, \\ w_t &= w_{t-1} + G(0, \sigma_w^2), \\ h_t &= h_{t-1} + G(0, \sigma_h^2), \end{aligned}$$

For which G is a Gaussian distribution and σ is chosen as $\sigma_x = \sigma_y = 6.4$ and $\sigma_w = \sigma_h = 0.64$[12].

Another very important part of the particle filtering is its observation model. This model defines how the weight of each particle is calculated. This is where the classifier C comes in for each particle. This is a continuous function that takes in a rectangle and an image and outputs a value between 0 and 1. This classifier can be of any type as long as it adheres to these input and output. In [12] it was proposed to use a modified Haar classifier[18], which splits the object in 4 pieces to make sure each section has enough features to be properly classified. If the particle is close to the target, the weight it is given is higher. If the particle is completely off target, the weight is close to 0. An example of the particle filter in action can be seen in Figure 2.

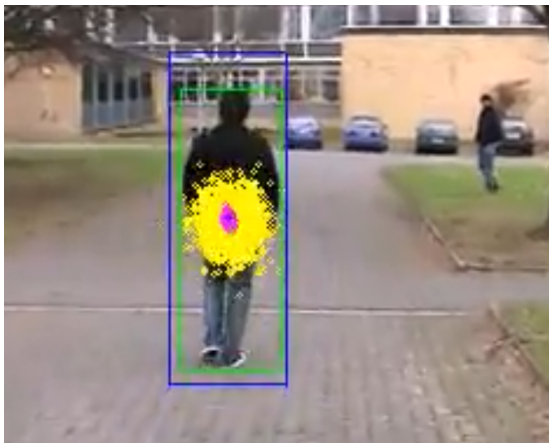

Figure 2: Particle filter tracking a person. The green box is the manually entered ground truth, the blue box is the resulting bounding box from the particles. The yellow particles have low weights, the purple particles have higher weights. Image originally from [12].

The intuitive idea behind a particle filter is that when enough particles are probing the image, those that are best at describing the target have the highest weights. Since all weights are calculated so they sum up to 1.0, if all bounding boxes from the particles and their weights are multiplied, the resulting bounding box should be a decent approximation of the target in the current frame. The idea of particles is based on natural selection, only the best particles in the current frame will survive and exist in the following frame. If a particle has a low weight, it has a low chance of surviving the current frame.

A disadvantage of the particle filtering is that it uses random sampling, which can be considered inefficient[9]. There are many more algorithms that need to run side by side for a robot to run effectively, so each of these algorithms have to be efficient enough for the whole to work. Every particle has to be evaluated and with 2000 particles (which is not uncommon[12]) this can be computationally expensive. As it is also based on RGB input, it is affected by illumination changes, as well as lack of illumination. Lastly the Haar classifier does adjust itself to new conditions, but if there is a sudden change then it might not adjust fast enough, decreasing its robustness.

4 Flood Fill on Depth

Since the introduction of Microsoft's Kinect[19], the interest in image processing using depth data has grown considerably. One of the biggest advantages of using depth data is that it is incredibly easy to distinguish objects from their background.

Using depth instead of RGB has a number of additional advantages. The only thing that can disturb the infra-red from the Kinect, is another infra-red source or see-through materials. The sun is one of the largest infra-red sources and can easily have an effect on the Kinect, however in a domestic environment there is often not enough sunlight to notice a big difference. Additionally, see-through materials should not affect a person tracker much, as persons are solid objects. Tracking purely on depth means you can ignore any RGB changes, so these algorithms are completely unaffected by changes in illumination or appearance. Someone could take off their coat and the algorithms will not be affected, as long as the person is still at the same depth, which is the property the tracker is looking for. Lastly

because the Kinect emits its own infra-red light and does not depend on additional light sources, the depth data is retrieved normally even when it is pitch dark, giving it another potential for rescue robots but also allowing it to function at night with limited light.

This section will describe the proposed alternative to tracking using RGB images: Flood Fill algorithm based on depth data (FFOD). FFOD was not the first attempted algorithm for tracking persons using depth data in this research; an alternative approach can be seen in appendix E.

FFOD depends on an initialization point or bounding box, which is supplied by the already existing face detector from the Robocup@Home team[3]. Whenever the robot is commanded to follow a person, it first needs to know where that person is currently located. In other words, it needs to initialize. For the evaluations that will be presented later, this has been handled manually. However in real life scenarios the robot should be able to do this completely autonomously. The way this is done at the moment is using the face detection system that is already incorporated in the robot. When someone tells the robot to follow him or her, the robot will detect that persons face and use that as initialization for the person tracking algorithm.

4.1 FFOD

The flood fill algorithm, as its name suggests, performs a flood fill using depth data as boundary. Instead of filtering the entire image, it plants a seed at the initial location of the person and from this point out it checks all direct neighbours. If the difference between the depth of the neighbour and the parent pixel is smaller than a certain threshold, the neighbour is added to a stack, waiting to be processed. See Figure 3 for an example of the flood fill algorithm.

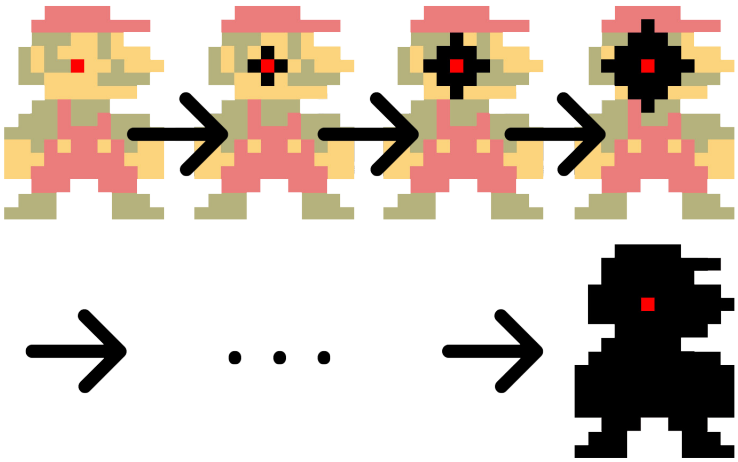

Figure 3: The effect of flood-fill on an image. The red pixel is the seed pixel, the black pixels are the 'flood'.

The threshold for the depth can be set really low for this algorithm, since the depth difference between two points belonging to the same object is usually not more than a few millimetres. While testing, this threshold has been set to 2cm. Setting the threshold too low will make the algorithm separate parts that actually belong to the same object, while setting the threshold too high will make the algorithm see different objects as one object more easily. Once the entire object has been filled, its center of mass is calculated and is used as initial point for the following frame. To avoid the problem of someone else stepping in front of the person and having

the algorithm seed that person from there on out, a check is done on the seed point. If the depth at the seed point drops from one frame to the other by more than a certain threshold, the seed is ignored until the correct depth is found again, our target is most likely occluded at this time. This threshold depends on the distance a person can move in one frame, the formula for this is :

$$T_d = v_p \cdot \frac{1}{fps}$$

Where T_d is the depth threshold, v_p the velocity of the tracked person and fps the frame rate at which images are received.

For a man in his twenties, v_p is equal to 2.53m/s[2], with a frame rate of 25 frames per second this means a threshold of approximately $0.1m$. Below is the pseudo code for the FFOD algorithm:

```
seedImage(seed, dt) :
  Queue q
  q.push(seed)
  while q not empty do
    if processed does not contain q.top() then
      if abs(q.top() − q.top().west()) < dt then
        q.push(q.top().west())
      if abs(q.top() − q.top().east()) < dt then
        q.push(q.top().east())
      if abs(q.top() − q.top().north()) < dt then
        q.push(q.top().north())
      if abs(q.top() − q.top().south()) < dt then
        q.push(q.top().south())
    processed.add(q.pop())
```

Next, based on the processed map, the average depth and center of mass can be calculated for the next frame.

Probably the biggest advantage of this algorithm over the other presented related work, is its simplicity and therefore also its efficiency. Most algorithms have to process an entire image at least once to be certain of the location of the target, however this algorithm only processes those pixels belonging to the same object. It will therefore almost never process an entire image, as the target rarely or never fills the entire frame. An example of this algorithm in action can be seen in Figure 4.

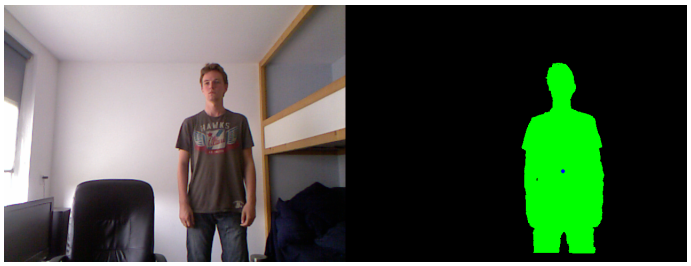

Figure 4: An example of flood fill algorithm in action. The green blob is the part that the tracker sees as target.

It does however also have some disadvantages. The first is that it assumes the center of mass lies on the person, but this is not necessarily the case, even if the

person was tracked properly. This can however easily be detected and then avoided (by for example choosing a random pixel that does lie on the person). Lastly, if the person touches anything with his body that is large enough to disturb the person tracking, the flood fill will 'escape' to this larger object, since it is focusing on the center of mass. For regular following through a room or corridor this is not something to take into account, but if the person has to open doors for example, extra modifications and checks are required. If the robot is far enough from the target person to also see the targets feet, then the ground is visually linked to the person, therefore the flood fill algorithm sees that as one object. This is not hard to bypass as well, by for instance setting some limitations to the width and height of the target. Most people are not taller than 2m and even if someone is taller than 2m, with a 2m boundary there is enough of the body selected to secure the effect of the tracker.

4.2 Comparison to related work

After showing the advantages and disadvantages of the FFOD algorithm, it is useful to compare it to the discussed related work in order to put it in perspective. This section will discuss how FFOD compares to each of the before presented algorithms. First a small comparison that holds for all of the mentioned algorithms, they are all based on RGB data. This means they are influenced, some more than others, by changes in illumination or appearance of the tracked person. FFOD works in an entirely different spectrum and is therefore a lot more robust against these visual changes, which is one of the biggest advantages of this algorithm.

As for Lucas Kanade specifically, it is very dependent on a system to keep feeding in new feature points to track. If this does not happen fast and good enough, the Lucas Kanade tracker has nothing to track and loses its target. Additionally if someone is wearing uniform coloured clothing, Lucas Kanade cannot properly keep track of its features, until it eventually loses its target. It is not uncommon for people to wear uniform coloured clothing, making this type of tracker unusable in these cases. FFOD is on the other hand perfectly capable of working on its own and is unaffected by the type of clothing of the target. The only case FFOD needs support is when something occluded the target for too long and that target has relatively moved. In this case FFOD is unable to regain the target and some other system should take over to reinitialize the tracker.

As mentioned before, OpenTLD is not performing good enough on malleable objects. It strongly depends on features to track and detect. If these features are constantly changing, then the tracker and detector loses its robustness. During the movement of a person, the features OpenTLD is looking for are constantly changing, while the feature FFOD is looking for, depth, only gradually changes, slow enough to easily keep track of the target. OpenTLD is only suitable as a person tracker as long as the target does not move its body too much, meaning that the features do not change too fast. This is not a constraint that should be demanded and therefore OpenTLD is not suitable as a person tracker.

While particle filtering has been shown to give very good results, it is inherently inefficient and complicated to get it to work properly. The simplicity of the FFOD algorithm compared to the particle filtering is a big advantage, while the results do not vary that much. With 2000 particles that need to be processed each frame, particle filtering is a lot less efficient than FFOD, which has to process only a small part of each frame once and can therefore easily keep up with the rate of the images. If all 2000 particles are processed, modern day systems would not be able to process this in real time, let alone when there are many other processes running on the system of a robot. If one of these processes causes a tremendous strain on the processing unit then the rest of the processes suffers from this effect as well.

Particle filtering is therefore considered to be computationally too expensive for modern systems.

5 Test suite

The evaluation of FFOD will consists of two parts. The first of which is that the MoSCoW requirements that were presented in chapter 2 are evaluated for FFOD. This will be done using created image sequences to isolate certain requirements and evaluate them independently.

The second part is to evaluate according to the Bonn Benchmark on Tracking (BoBoT). At the time of writing, the BoBoT data set was just expanded to include depth data in their sequences. It is this BoBoT-D data set which FFOD will be benchmarked on. This section will start with evaluating the MoSCoW requirements, after that the BoBoT dataset will be used to evaluate the algorithm compared to a state of the art person tracker.

5.1 MoSCoW requirements evaluation

In order to evaluate the quality of FFOD this paper will evaluate the MoSCoW requirements that were set up in chapter 2. This paper will put the main focus on evaluating the Must have and Should have requirements as these are the most important requirements. The Could and Won't have requirements are additional features that make the implementation better, but the algorithm can be considered successful without these requirements fulfilled, therefore little attention will be paid to these last two types of requirements.

In order to evaluate FFOD, a number of image sequences have been created to put several requirements to the test. This evaluation is not to test the precision of FFOD, but rather to test how many requirements it fulfills. Therefore, the recorded image sequences do not contain a ground truth bounding box, which can be used to measure precision. Instead, sequences have been created to test separate requirements. These sequences have been tested in five scenes, which will be explained first. The scenes are :

1. A fairly sterile room with a white background. This scene is considered to be a somewhat simple surrounding. The background is not very cluttered and there is enough light coming from the side. This scene is considered to be a basic and easy scene for algorithms.

2. A cluttered living room with yellow furniture and with sunlight shining towards the camera. This scene has a much more cluttered background. This makes it more difficult to properly detect and track a person, because algorithms can more easily mistake background for the target object. Additionally the light makes the person a lot darker and therefore the camera loses information on its appearance.

3. A white kitchen with little room to walk in. The narrow room could cause confusion for trackers based on distances between found objects of interest. Additionally the background is also rather cluttered, but in this case the person stands out more from the background than with scene 2.

4. Same environment as scene 1, but recorded at night with very limited light. When there is little information known about the environment it is commonly hard for trackers to find feature points on which to focus. Additionally because it is dark, everything in the scene starts to look similar to one another.

Examples of these scenes can be seen in Figure 5.

Figure 5: A view of one of the image sequences for each environment.

Additionally there is one extra scene to test requirements in which a person walks forward for approximately 10 seconds. This was not possible in the before mentioned scenes due to a lack of space.

5. A long straight hallway with rooms on both sides, blue carpet and white walls. There are some posters and boards on the walls. The person blends in with the scene quite well.

These scenes are intended to test a variety of evironmental factors. The five image sequences that are tested in the first four scenes are :

1. The target person is standing still for 10 seconds. This tests requirement 1.2.
2. The target person is rotating and later moving while rotating. This tests requirements 2.3 and 2.4.
3. The target person is randomly walking in front of the camera. This tests requirement 1.3.
4. The target person is walking away and towards the camera. This tests requirement 1.5.
5. The target person is standing still while putting on a coat or removing a coat. This tests requirement 3.2.

Each of these sequences is tested in each of the scenes, as to test the effect of each scene on the requirements.

Another image sequence been recorded in a hallway to test requirement 1.4, for which an example can be seen in Figure 6. This sequence has been recorded in the fifth scene, the long and straight hallway.

6. The target person is walking for approximately 10 seconds through a straight hallway. This tests requirements 1.4 and after omitting some frames, 2.2.

For requirements 2.1 and 3.3 the image sequence of the BoBoT benchmark was used, it is a good representation of a real life scenario of people walking in between and creating a separate sequence was therefore not necessary.

7. The target person is walking through a straight hallway. During this time there are three people walking in between the target and the robot at random intervals. This tests requirement 2.1 and 3.3.

Additionally, requirement 1.1 and 1.6 can be tested on any sequence, however for these requirements the sequence itself does not matter. For requirement 1.1 only the first frame needs to be tested, which is almost identical in all sequences. Requirement 1.6 can be tested as long as the tracker is actively tracking. Therefore

these two requirements are only tested once per scene, in order to test the environmental factors only. For the results later on, they were tested on sequence 1, but any of the sequences could have been selected.

Figure 6: An image of the walking image sequence.

In table 5.1 each requirement is displayed, categorized by importance. The third column displays the sequence number with which the requirement was tested. The last six columns show the result for each of the used scenes (S1 - S5 and the BoBoT sequence). If a requirement was not tested in a certain scene, its cell is coloured gray. A sequence is considered successful if the tracker initialized and the person was not lost during the sequence. This includes that no additional objects or elements in the scene should be considered target (with the exception of the coat for the coat sequence).

	Must have	Seq	S1	S2	S3	S4	S5	B
1.1	Successfully find a bounding box in initial frame	1-7	✓	✓	✓	✓	✓	✓
1.2	Stay on target for 10s while standing still	1	✓	✓	✓	✓		
1.3	Stay on target while target is moving within 5m of robot	3	✓	✗	✗	✓		
1.4	Stay on target while following for 10s	6					✓	
1.5	Stay on target while person is walking away	4	✓	✗	✗	✓		
1.6	Maintain a frame rate of at least 20fps[a]	1-7	✓	✓	✓	✓	✓	✓
	Should have							
2.1	Pause or keep track of target when someone walks in between[b]	7						✓
2.2	Stay on target while frame rate drops to 10 frames per second[c]	6					✓	
2.3	When the person is rotating, the tracker should stay on target	2	✓	✓	✓	✓		
2.4	When the person is rotating while moving, the tracker should stay on target	2	✓	✓	✓	✓		
2.5	When the person is lost, the tracker should regain its target when possible	-						
	Could have							
3.1	If the person takes a corner, the tracker should regain when the robot takes the same corner	-						
3.2	If the person puts on a coat, the tracker shouldn't be lost	5	✓	✓	✓	✓		
3.3	If an identical person walks in between, the original target should still be tracked[d]	7						✓
3.4	If the target is temporarily completely occluded, the tracker should estimate the persons position	-						
	Won't have							
4.1	If the person completely disappears in a crowd, the tracker will pause until the target is visible again	-						
4.2	If there are multiple identical people walking in the screen, the tracker should properly stay on target	-						

[a] An average computation time of 5 milliseconds was measured during testing, which means a potential to keep up with 200 frames per second.

[b] Result is based on the image sequence from the BoBoT benchmark, where 3 people walk in between. Only works if the person and the robot keep the same distance.

[c] Tested on the walking sequence, where some frames were omitted. Depends on the speed of the target whether it works or not, test was executed at roughly 1.0m/s

[d] Requirement is for FFOD the same as 2.1, as appearances does not matter in depth data

Table 1: Results from the requirements evaluation.

All the Must have requirements successfully passed the requirements, only 1.3 and 1.5 were having some problems with the image sequences in the living room

and in the kitchen. The reason for this is that there was a lot of space to move back and forth and when the distance was large enough, the camera could see the feet being connected to the floor. In all 4 sequences this caused the flood fill to 'escape' and lose the person. Rerunning the test with a limited width and height for a person resolved the issue, so this is not considered a big problem. However this is something that needs to be worked on in the future. FFOD is only bounded by the depth of the pixels, but perhaps some additional boundary can be defined.

Most of the Should have requirements have also been fulfilled, except for the last one, 2.5. The reason for this is that FFOD is missing an observation model, some system to regain the target if the tracker has been lost. For the time being this was not a crucial feature, but this is something that should be looked at in the future as well. Because this feature is missing, requirements 3.1 and 4.1 are also considered not implemented. 3.4 is not implemented because this algorithm is not using a probabilistic method to estimate a persons position, it is basing its result purely on what it sees. Estimating a persons position requires some sort of probabilistic algorithm.

5.2 BoBoT-D benchmark

As a second part of the evaluation, FFOD will be benchmarked using the Bonn Benchmark on Testing (BoBoT) dataset. It has been only recently that a second type of benchmark was added, the so called BoBoT-D dataset. This dataset also contains depth data for each sequence, which is what FFOD will be benchmarked on. In order to compare precision of person tracking algorithms with each other using the BoBoT-D benchmark, the ground truth bounding box was defined. This is a manually defined bounding box for each frame of the dataset, this box should be a closest fit around the person to be tracked. Algorithms can then output their bounding boxes of the tracked target and an overlap between the ground truth and the algorithms bounding boxes can be calculated. If the bounding box exactly covers the ground truth, there is a 100% overlap. The precision of an algorithm is considered the average of these overlap percentages from all frames in a sequence. An example of the overlap can be seen in Figure 7.

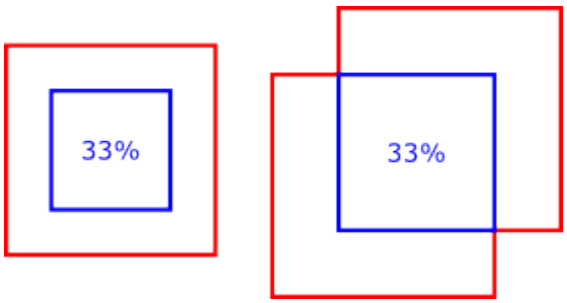

Figure 7: An example of how the overlap for bounding boxes is calculated in the BoBoT dataset. The blue box represents the overlap of two bounding boxes. Image taken from [4].

There are as of now only a few sequences in the BoBoT-D dataset, among them there is one sequence of a person moving through a hallway where people walk between the robot and the target person to cause full occlusions. Near the end of the sequence there is also a shake introduced in the camera, to simulate the robot driving over some rough terrain. An image of the sequence can be seen in Figure 8.

Figure 8: A still of the person following sequence from the BoBoT-D dataset. Image taken from [4].

As of now there is one algorithm which is benchmarked using this dataset due to the benchmarks novelty, which is a particle filter algorithm[5]. This algorithm can be considered as a state of the art person following algorithm. A summary of results can be seen in table 5.2. The precision of the particle filter tracker on the person following sequence in the BoBoT-D dataset was **71.92%**[5]. The FFOD algorithm scored almost 10% higher, at **80.09%**. The paper which discusses the particle filtering also measured the hit percentage, which was defined as the number of times the overlap is higher than $\frac{1}{3}$[5]. The hit percentage for their algorithm was at best **95.92%**, while the FFOD algorithm scores marginally higher, with **96.04%**. Because the hit percentage is more or less equal, both algorithms were tracking the object an almost equal amount of frames, however the FFOD had a tighter fit around the target due to the nature of flood fill algorithms.

	Precision	Hit percentage
Particle Filtering	**71.92%**	**95.92%**
FFOD	**80.09%**	**96.04%**

Table 2: Results from the BoBoT data.

6 Discussion

It can be argued that the overlap between the ground truth bounding box and the resulting bounding box is not a good performance measure. With the current definition of overlap it can occur that the resulting bounding box is perfectly placed around the person but too big, as with the left example in Figure 7. This would seem more desirable than a resulting bounding box that is equally big as the ground truth, but shifted off target. However as can be seen in Figure 7, both can result in the same overlap percentage. A better measure would be to take the center of the resulting bounding box and the center of the ground truth into account as well as its size. In this case a higher importance can be given to having the center be correctly on the target, which is for person tracking the more desirable feature. This would also bring the precision of FFOD and the particle filter closer together, as the particle filter algorithm seems to, based on published video's, create a rather wide bounding box. The penalty for this somewhat wider bounding box would then be less severe compared to how it has been measured before.

A similar issue occurs with bounding boxes that are too big or too small. It is more favorable to have a bounding box too tight around a person, than a bounding

box that is too wide. In the measurement of BoBoT this does not matter and can result in an equal overlap. It is however even more favorable to have the bounding box placed exactly on the person. To summarize, different levels of importance can be identified, however these are not handled in the BoBoT data set.

There is also a flaw in measuring the precision of an algorithm in the BoBoT data set. If the target is fully occluded, then the ground truth will say there is no bounding box. If the algorithm also says there is no bounding box, this should mean an overlap of 100% since it is saying the exact same thing as the ground truth bounding box. However, the evaluator tool calculates this as a 0% overlap. This can be easily changed, but was not done for the benchmarking done here, because then the results of the particle filter and FFOD would not be comparable. Measuring a correctly detected occlusion as a 100% overlap would have resulted in a precision of 83.3% instead of 80.09% for FFOD.

Lastly, FFOD has a very basic way of regaining the person when it is lost. It tries to find the person in a small area from where the person was last found. This works good enough when someone quickly walks between the robot and the target person, but it is in no way a quality method for regaining the person. The BoBoT dataset does not exclude such simple regaining however, therefore FFOD managed to score highly on their benchmark. Had it included some sequence to make sure this method of regaining a target was not possible, FFOD would have scored significantly lower. In this case the particle filter algorithm would have scored a lot higher in comparison with FFOD.

7 Future work

As was stated in section 4.1, FFOD needs some more work to be more robust. There are a couple of scenarios and weak parts of the algorithm that can be improved. The most important one is the regaining of a target person. Many trackers keep updating a detection model, so that when the tracker sees that it has lost the target, it will let the detection model take over until this model can reinitialize the tracker. This was however not implemented in the current version of FFOD because of time constraints, however it would make it that much more robust.

Another point of attention would be to constrain the flood fill better. At the moment it is purely constrained by the depth difference between the neighbour pixel and its parent. This can mean that if the person is touching something or if the robot sees the feet of the person on the ground, that the flood fill 'escapes' to unwanted parts of the frame. One of the more simple solutions would be to limit the size of the flood fill by the average size of a person, which has been briefly tested and did resolve some issues. However if a detection model is present, then perhaps other constraints using this model can be made.

Lastly it could be interesting to add some motion model to the tracker, so the tracker can estimate the targets position while it is not seeing the target. This will let the tracker, and perhaps the observation model, more easily regain the target whenever it becomes visible again.

8 Conclusion

This paper introduced the problem of following a person in a domestic environment and showed related work using RGB images that was already done for this type of application. An alternative approach was introduced, Flood Fill on Depth (FFOD), using depth data from Microsoft's Kinect. FFOD was evaluated by checking the requirements introduced in this paper, thereby solving the initial problem

statement. FFOD was additionally benchmarked using the Bonn Benchmark on Tracking (BoBoT-D) dataset. According to this benchmark, FFOD had a near 10% higher precision on the followed person than a state of the art particle filter tracker.

There are still flaws in the FFOD algorithm however, flaws that need to be addressed before it can be used reliably. The most important missing feature is a detector that allows the tracker to regain its target when lost. The second missing feature is some additional boundary to the flood fill, to prevent it from filling non-person points.

Based on the found results the conclusion is that there is a good future for person following using depth data, as FFOD has shown. A relatively simple algorithm, using depth data, was able to compete against a state of the art person tracker. During the research stage there was hardly any publications found on this type of application using depth data, while there are good opportunities to use features from depth data to design very robust trackers. Perhaps the best results can be obtained by merging depth data with existing trackers that have proven to function good and efficiently on RGB data.

A Orientation report

The orientation report was constructed in the first few weeks. It is intended to give some knowledge about related work that is already done on the subject at hand. It also defines the estimated approach for the project, like the used software tools and the environment I worked in. Parts of the report was reused for the final report.

A.1 Introduction

This report will present the orientation phase of my project that will be performed as part of the bachelors project at the TUDelft, specifically the Robocup@Home team at the faculty for mechanical engineering. This project will focus mainly on the image processing part for allowing a robot to follow persons around a cluttered domestic environment in real time. In this report I will discuss the requirements for such a feature using the MoSCoW model and I will discuss the environment I will be working in. Since there are already a number of papers written about tracking persons, this report will also talk about some of these algorithms and how they can be used during the project. Based on the found literature, a new algorithm will be constructed which should fulfil the most important requirements. At the end of the project the feature for following persons should be implemented on the Robocup@Home robot called Robby, so that Robby can complete the FollowMe challenge in the Robocup@Home competition.

A.2 Environment

This section will describe the environment this project will take place in. It will discuss the people I will be working with, the area I will be working in and the IDE and software tools I will be using.

A.2.1 Colleagues

During my project I was told I will mostly be working with three people; Maja Rudinac, Aswin Chandarr and Floris Gaisser. Maja is also my project coordinator at the faculty of 3mE. Aswin is a master student in BioRobotics and is mostly working with vision systems on robots. Floris is a master student at the department of Bio-Mechanical Design and is mostly working with object classification and face recognition systems. All three of these people have a lot of experience with vision systems in robotic systems.

A.2.2 Workplace

The area I will be working in will mostly be in the Robocup@Home room, which is in the robot lab of the mechanical engineering faculty (3mE) at the Technical University in Delft. This robot lab is home to several robotic project, of which one is the Robocup@Home. This room is partially setup to reflect a household area, the rest is a normal working area. Most of the times there are a couple of people from the Robocup@Home team present there, working on the robot or doing research. There are no pre-determined places so everyone sits and works where there is room (with the exception of some desktop PCs that are present).

A.2.3 Software tools

I will mostly be working with existing software packets that the Robocup@Home team are already using, to make the integration with the robot easier. Luckily the

Robocup@Home team is using the same software tools I was using during my minor, so I had enough experience with these tools. The robot itself runs using the Robot Operating System, or ROS. ROS is an open source framework, allowing easy communication between processes, the so called nodes, and it supplies many necessary tools and packets for controlling a robot, such as navigation, motor control and even some vision packets. Another software packet I will use is a commonly used package for image processing, OpenCV (Open Computer Vision). OpenCV allows for easy handling of incoming images in the form of matrices and it also provides many possible operations on these matrices. From low level adding and subtracting to higher level algorithms like contour detection or integral image calculations. Therefore OpenCV is very useful when doing any type of image processing. The company behind OpenCV is also the company behind ROS, so there is a nice integration between these two libraries.

All these tools are based on, or have wrappers for, the C++ programming language. The existing code from the Robocup@Home competition is therefore also in C++. As for the IDE I will be using, there were not many options. I will have to work in Ubuntu for starters, since ROS runs best on this operating system, and for Linux there aren't that many C++ IDE's. The most known IDE is probably Eclipse, with the CDT plugin to allow for C++ programming and compilation. Luckily ROS supports Eclipse projects as well, by supplying each packet you make with a script that can generate Eclipse project files. Additionally I have enough experience through courses from my study with Eclipse.

A.3 Literature

This is not the first attempt to create a person tracker, over the last two decades there have been many proposed and designed algorithms to track persons. Some of these are designed for automated surveillance, while others are designed for use in robotics. Those that are used in surveillance could have some different requirements. They often need to be capable of tracking more than one persons at a time and the cameras are usually positioned in high places. Because the cameras are stationary, this implies they have a static angle and an almost static background. Because of these differences it is more interesting to look into algorithms designed for robots specifically.

A.3.1 Lucas Kanade

Lucas Kanade is often used to track objects in regular RGB images. It is however just a tracker, in the sense that if an object is lost, by say occlusion, the tracker will not re-detect the object when it appears again. That is why the Lucas Kanade tracker is usually combined with a detector. This detector detects parts of an image that are of interest, possibly the target object. The computationally more efficient tracker then takes over and tracks this region until it is lost. When that happens, the detector tries to locate it again and the cycle repeats. It is common to have this detector be updated based on the result of the tracker, so that changes in the scene do not affect the detector as much as with a static detector.

The Lucas Kanade tracker is a type of optical flow tracker, this means the algorithm tries to match a previously recorded image with the currently received image. By finding this transformation between the two images, you can transform certain feature points. A feature point is basically a point which can easily be recognized from the background, like the nose or eyes of in a face. It is these key points in an image that it tries to track from one frame to the other, thereby tracking the object.

A.3.2 OpenTLD

OpenTLD, originally named Predator, is a combination of **T**racking, **L**earning and **D**etecting. The algorithm is very new, its source code was released in 2011. Originally created in Matlab, there are now many C++ ports available. One of these ports was created as part of somebody else's master thesis. It runs much faster than the Matlab implementation and performs almost identical. The reason this specific port is interesting, is because there is enough documentation available for this code, and the creator properly subdivided the different parts of the algorithm in different classes. Other ports are generally a direct copy of the Matlab code, which ends up becoming unreadable and hard to work with.

OpenTLD uses a modified Lucas Kanade tracker for the tracking part. The tracker is initialized with a bounding box of the current target. At that time, the detector is also initialized. For the following frames, the tracker tries to keep up with the objects motion and predict its position. The resulting bounding box from the tracker in turn keeps the detector updated, so that any changes in appearance of the object are properly adjusted in the detection model.

Might the tracker lose the object, when it goes outside of the screen, or if it gets occluded or when it is simply moving too fast, the detector takes over again and re-initializes the tracker when it has detected the object. Although not specifically designed for tracking persons, this algorithm is supposed to be capable of tracking almost any type of object, so I think it is worth looking into for a person tracker.

A.3.3 Particle Filter

The particle filter tracking method is somewhat newer than the Lucas Kanade tracking. It is based on the condensation (**con**ditional **dens**ity propag**ation**) algorithm, which was first introduced in 1998. The condensation algorithm is a probabilistic algorithm[9], which was designed to robustly track agile moving objects in a cluttered environment. A particle filter tracker exists of particles that are acting as a sort of probes on the image to detect the object. These particles can be modelled as a vector[12]:

$$\mathbf{x} = (x, y, w, h, v_x, v_y, C)^T$$

For which $\mathbf{x}$ is the particle, (x, y, w, h) the bounding box describing the object to be tracked, v_x and v_y its speed in x and y respectively and lastly C its classifier. Each particle is moving randomly on the image, based on the so called motion model, and its bounding box is randomly adjusting its size, based on a Gaussian distribution. The motion model used in this algorithm is given as follows[12]:

$$\begin{aligned}
v_{x,t} &= v_{x,t-1} + G(0, \sigma_x^2), \\
v_{y,t} &= v_{y,t-1} + G(0, \sigma_y^2), \\
x_t &= x_{t-1} + v_{x,t}, \\
y_t &= y_{t-1} + v_{y,t}, \\
w_t &= w_{t-1} + G(0, \sigma_w^2), \\
h_t &= h_{t-1} + G(0, \sigma_h^2),
\end{aligned}$$

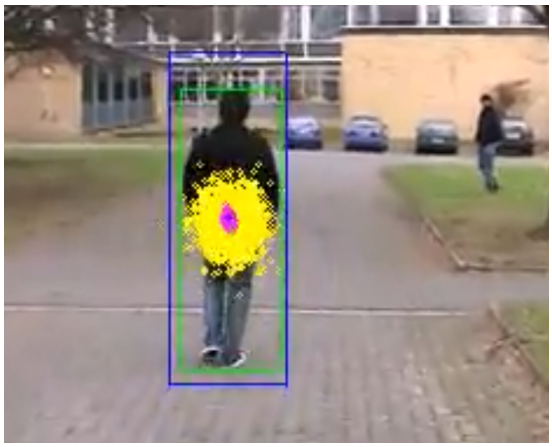

Figure 9: Particle filter tracking a person. The green box is the manually entered ground truth, the blue box is the resulting bounding box from the particles. The yellow particles have low weights, the purple particles have higher weights. Image originally from [12].

For which G is a Gaussian distribution and the σ_i is chosen as $\sigma_x = \sigma_y = 6.4$ and $\sigma_w = \sigma_h = 0.64$[12].

Another very important part of the particle filtering is its observation model. This model defines how the weights of each particle should be. This is where the classifier C comes in for each particle. This is a continuous function that takes in a rectangle and an image and outputs a value between 0 and 1. This classifier can be of any type as long as it adheres to these input and output. In [12] it was proposed to use a modified Haar classifier[18], which splits the object in 4 pieces to make sure each section has enough features to be properly classified. If the particle is close to the target, the weight it is given is higher. If the particle is completely off target, the weight is close to 0. An example of the particle filter in action can be seen in Figure 9.

The intuitive idea behind a particle filter is that when enough particles are floating around in the image, those that are best at describing the target have the highest weights. Therefore if all bounding boxes from the particles and their weights are multiplied you should receive a decent approximation of your target in the current frame. The idea is based on natural selection, only the best particles in the current frame will survive and exist in the following frame. If a particle has a low weight, it has a low chance of surviving the current frame.

B Task description

This appendix will introduce the assignment, as well as define the requirements for the algorithm. These requirements are also used in the paper to evaluate the algorithm for a successful implementation. This appendix also contains the official task description of the Robocup@Home competition for the Follow Me task, as well as the bachelors project task description that was posted on Blackboard by the Robocup@Home team.

B.1 Introduction

My interest in this assignment began when browsing through all the bachelor tasks on the Blackboard page. I came across an assignment from the Robocup@Home team at the Technical University of Delft, this assignment can be seen in section B.3. I was already a little familiar with this team as I had met some of them during my minor robotics at the same institute. The assignment describes four students for which they had room. I was mainly interested in the task of student three and four, which mention image processing. I have been interested in image processing for a few years already and I saw this as an excellent opportunity to increase my knowledge in this field.

B.1.1 Faculty description

The Delft Robotics institute is working on a variety of projects, such as the walking robot TUlip, which is a prototype for the soccer robot. A subdivision of this institute is the Robocup@Home team. This team consists of roughly a dozen bachelor-, master- and PhD-students. These students have been working on robot Robby for a year now and are competing in the Robocup@Home 2012 competition, which is taking place in Mexico in June. The goal of the team is to make an 'affordable personal robot that can be employed in any domestic or care environment'[17]. In order to compete in the Robocup@Home competition there are a number of features the robot has to be able to perform. One of these tasks is the 'Follow Me' task, for which the description can be found in section B.4.

For this Follow Me task, a person tracker was needed and I was given the task to make such a tracker. The tracker should be fast enough to work alongside the other processes running on the laptop that controls the robot and should also keep track of the operator (as defined in the Robocup@Home task) in each frame if he is present.

B.2 Task description

This chapter will go deeper into the actual description of the task that had to be completed. It will start with the exact task description and finish with the criteria for determining a successful implementation.

The Robocup@Home team already created algorithms for detecting and locating persons, after a person has been located he should however be tracked, as the Follow Me task describes. The requirements for the Follow Me task can be subdivided using the MoSCoW (must have, should have, could have and won't have) model;

B.2.1 Must have

Must have requirements are requirements that the tracker must atleast fulfil before a method is considered successful. In this specific case the requirements are designed so that if all are met, the robot can do the most simplest of person tracking for a short amount of time.

Given an initial point or bounding box of a person, the tracker must ...

- ... successfully find a bounding box in the same frame of this person.
- ... not drift off target and keep track of the person for at least 10 seconds, while the scene changes are ignored. During this time the person is standing still.
- ... keep track of the person for 10 seconds while the person is moving forward, backward or sideways in front of the robot in an area a maximum of 5 meters away from the robot. During this time the robot will stand still and the person is not visually obstructed and facing the same direction as with initializing.
- ... keep track of the person for 10 seconds while the person is moving forward at a normal speed for robots, roughly 1m/s, and the robot is following at the same speed (the relative speed between the robot and the person is in this case 0m/s).
- ... keep track of the person while the person is walking away and within 5 meters of the front of the robot. During this time the robot is standing still.
- ... keep a frame rate of at least 20 frames per second while tracking on modern day systems[2] with a frame rate of 25-30 frames per second.

B.2.2 Should have

Should have requirements are requirements with high priority for the algorithm. These are generally very important requirements, but not critical ones. The algorithm can still function somewhat effectively if not all or none of these requirements are met.

- When both the robot and the tracked person are standing still and someone with a different color clothing is walking between the two, the tracker should either momentarily pause or keep the bounding box on the correct target and resume normally when the target is visible again.
- The tracker must keep track of the target for 10 seconds with any frame rate between 10-30 frames per second, while both are moving forward at the same speed.
- When the person turns around in place, the tracker should not lose the person.
- When the person turns around while moving, the tracker should not lose the person.
- If the tracked person for any reason is temporarily lost and reappears in the screen, the tracker should regain its target.

B.2.3 Could have

Could have requirements are considered desirable, but not necessary for the effectiveness of the algorithm. If enough time and resources are available, it is generally worth while to have a look at these requirements.

- If the person moves around a corner and is visually obstructed by the wall, the tracker should regain the person after it has cleared the corner and sees the person again.

[2]For the benchmark and additional tests later on, an Intel system with dual core running at 2.0Ghz is used.

- If the person changes appearance during tracking, by for instance putting on a coat, the tracker should not lose the person.
- If the person and the robot are standing still and someone who looks almost identical to the tracked person walks in between, the robot should afterwards still be tracking the original person.
- When the person is temporarily completely occluded, the tracker should estimate where the person will be.

B.2.4 Won't have

Won't have requirements are generally not implemented in the current release. They are requirements to which can be looked at in the future.

- If the person completely disappears in a crowd of people, the tracker will pause until the to be tracked person reappears from the crowd.
- If multiple persons are walking in the view of the robot and they all look more or less identical, the robot should still keep track of the person it was told to keep track of, regardless of the motion of the persons.

B.2.5 Criteria for successful result

Now that the requirements are established, defining criteria for a successful implementation becomes easy. For a successful implementation of a person tracker, the tracker should atleast pass the must have requirements.

Passing some, or perhaps all, 'should have' requirements is expected but not required for a successful implementation. Passing any other requirement from the list of requirements is an added bonus, but not a necessity.

B.3 The assignment

Title of project proposal:
Software architecture for affordable personal robots
Keywords:
Task coordination framework, Testing and validation, Object recognition and motion analysis, Safe human robot interaction
Introduction:
Bringing personal robots to households requires solving many challenges. The robots need to deal with the unstructured and dynamic environment, need to safely interact with humans and to be very affordable for the large market of consumers. In the BioRobotics lab, we have constituted Robocup@Home team of students with a goal to design the first Dutch affordable service robot. From this year we will also participate in the RoboCup@Home league competition, which is the largest competition of service robots, where they need to perform complex household tasks such as: interaction with humans, bringing and preparing food, cleaning the items from floor, shopping in the mall, etc. The team is consisting of both masters and bachelor students from both 3ME and EWI who are designing and implementing all required algorithms and preparing the robot for the competition.

Service robots must be capable of performing activities such as object recognition and grasping, autonomous navigation and safely interact with their respective environment. Each of these activities requires complex algorithms to be executed on a computer and subsequently communicate with the robot hardware to achieve the desired actions. These algorithms constitute the Brain of the robot, You will have the opportunity to work in the multidisciplinary team and to test your algorithms on the real platforms DPR2 and Tuxedo. And of course to help us bring personal robots to the every household.

Research questions:

- Student 1: Implement a task coordination framework for Tuxedo such that it meets the real-time task demands of a service robot. The framework has to be generic enough to be portable to different service robots.
- Student 2: Design and implement guidelines for the verification and ratification of the Tuxedo software, based on the existing testing frameworks at both a component and functional level. Student 1 and Student 2 will closely collaborate together.
- Student 3: Design of a real time object recognition module for a mobile robot in dynamic and cluttered environment combining 2D and 3D information available from different sensors.
- Student 4: Design of a motion analysis module for a mobile robot including multi sensory calibration, ego motion estimation and compensation and tracking of rigid objects and people using combination of 2D and 3D sensors. Student 3 and Student 4 will closely collaborate together.

All students: All modules will be applied and tested on the low cost personal robot Tuxedo. Requirements:

1. Programming skills: Knowledge of C++ is required
2. Experience on Robot Operating System (ROS) and Linux is a plus
3. Finished minor in robotics is a plus but not required

Remarks:
Students will need to collaborate together and with the multidisciplinary Robocup@Home team. Results of the bachelor theses will be used for the Robocup@home competition in June this year.

Supervisors:
Maja Rudinac and Dr Wouter Caarls
Biorobotics Lab, 3ME
TU Delft

B.4 RoboCup@Home Follow Me task

The robot has to safely follow an unknown person through a public space.

B.4.1 Focus

This test focuses on tracking and recognizing a previously unknown person, basic interaction and signalling capabilities, and safe navigation in unknown environments and narrow spaces with other people walking around or blocking the way.

B.4.2 Setup

1. Location: The test takes place outside the arena in a public space.
2. Operator: A professional operator is selected by the TC to test the robot.
3. Other people: There are no restrictions on other people walking by or standing around throughout the complete task.
4. Path: A path is setup (but not announced) beforehand. The complete path is divided into three sections by two intermediate time points (ITPs).

B.4.3 Task

1. Start: The robot starts at a designated starting position, and waits for the professional operator. When the referees start the time, the team is allowed to (briefly) instruct the operator. After the instruction, the operator steps in front of the robot and tells it to follow (no start signal).
2. Memorizing the operator: The robot has to memorize the operator. During this phase, the robot may instruct the operator to follow a certain setup procedure.
3. Following the operator: When the robot signals that it is ready to start, the operator is walking in a natural way on the designated path. The robot needs to follow the operator. The robot deals with different obstacles (single persons, tight elevator rooms, and small crowds) in different sections. Each section has a separate time measurement, but the complete task needs to be performed within the overall task time. 1st section (from start to first ITP): Two persons block the direct passage on the way to the first ITP (at different positions). The operator guides the robot around the persons. One of the persons starts walking when the robot approaches, and crosses the way between robot and guide. Right in between, the walking person stops for 3 s before walking away. 2nd section (between first and second ITP): The operator guides the robot into a small tight room (e.g., an elevator). There is not enough space to freely operate in this room (e.g., 1m x 2 m). Other people may already be in there causing that operator and robot have to leave in reverse order (of entering). The team may choose among the following options and instruct the operator accordingly when the test starts. RoboCup@Home Rulebook / Final version for RoboCup 2012, (Revision: 286:288)

   - Either the operator enters first, and sends out the robot for leaving, or
   - the operator sends in the robot first, and exits first when leaving.

   Between entering and leaving the room, the door is shut for at least 5 s. In case the room is not directly accessible, the time is stopped while waiting. 3rd section (between second ITP and finish line): A small crowd of people (4-5)

will be waiting outside the room, blocking the way between the second ITP and the finish line. The operator sneaks through the crowd and waits for the robot on the other side. The robot cannot pass through the people and needs to autonomously navigate around the group. While waiting, the operator is allowed to signal the robot where it is standing (e.g., by waving), but it is not allowed to move back. After the robot is following the operator again, the operator proceeds to the finish line.

B.4.4 Additional rules and remarks

1. Preparation: The robot needs to wait for at least 1 min before the operator appears in front of the robot. During this waiting time the team is not allowed to touch the robot.

2. Natural walking: The operator has to walk naturally, i.e., move forward facing for- ward. If not mentioned otherwise, the operator is not allowed to walk back, stand still, signal the robot or follow some re-calibration procedure.

3. Asking for passage: The robot is allowed to (gently) ask individual persons to step aside, but it is not allowed to blindly shout at groups of people.

4. Disturbances from outside: If a person from the audience (severely) interferes with the robot in a way that makes it impossible to solve the task, the team may repeat the test immediately.

5. Instruction: The robot interacts with the operator, not the team. That is, the team is only allowed to (very!) briefly instruct the operator

   - how to the tell the robot to follow,
   - how to signal it (e.g., waving), and
   - how to get it into or out of the tight room (ITP 2).

6. Calling the operator back: When the robot has lost the operator, it may call the operator back once per section (loosing the points for that section).

B.4.5 OC and Referee instructions

Any time before the test:

- Define the path and ITPs where the time is measured.
- Select the professional operator(s).

During the test:

- Show the path and the ITPs to the operators.
- Take the (accumulated) time at each ITP and at the finish line.
- Check that the elevator door is closed for the specified time. RoboCup@Home Rulebook / Final version for RoboCup 2012, (Revision: 286:288) 36 4.2 Follow Me
- Check save operation of the robot; the robot needs to be stopped immediately if a person is going to be touched by the robot

C Teamwork

Officially I am the only one working on this bachelors project, however I did not have to do everything by my self. At the orientation phase of my project I was guided a lot by my 3mE supervisor, Maja Rudinac. At the time I was not aware of all the algorithms that could be used for such a task and I relied heavily on her helping me and answering my questions. At this time we were discussing papers and possible approaches together with Xin Wang to figure out about how to handle this project. Some of these approaches, such as particle filtering, ended up in my paper as well.

After orientation, the idea was that we would first replicate the particle filter from [12] and built on from that, perhaps extend it using depth data from the Kinect. Maja was really occupied with getting everything organized for the trip to the competition in Mexico, so at this phase she was only partially available. Luckily there were two skilled people, Aswin Chandarr and Floris Gaisser (no family), almost constantly available in the robotlab to discuss algorithms with and help with coding. The three of us were working as a team at first on focusing to get the particle filtering to work, with some help from Wouter Caarls and Maja Rudinac. After a while we ran into some problems with the particle filtering which we couldn't resolve in time, I came up with the suggestion of tracking a person based purely on depth data and discarding the particle filter. At first the general opinion was that it probably would not work good enough, but the algorithms I had in mind were not hard and a simple implementation could be set up in a day or two to test it out. After discussing what would be the best methods to do this, we decided it would be best if we each tried our own implementation and then discussed later on what we found. Aswin was trying to get the particle filtering to work, Floris was working on the algorithm that is described as the depth threshold algorithm in my paper and I was working on the flood fill on depth. After Floris and I were done we started comparing the results and found some flaws in the threshold algorithm, as was described in my paper. The flood fill algorithm appeared to be very promising however, so we decided to drop the progress on the particle filtering and the threshold algorithm and continue with what would later be known as FFOD.

After having finished the FFOD algorithm it was time to put it to the test on the robot. Aswin had already built some code that would translate the position of a person into a velocity for the robot. This translation wasn't working 100% yet, so Aswin, Floris and I spent the rest of the day tweaking this code. Eventually we had everything working and the robot was following us at fairly high speeds, up to 1.4m/s. The following weekend the Robocup@Home team went to Eindhoven, where they could immediately stress-test the system. An image of this informal test can be seen in Figure 10.

Figure 10: First demo of the FFOD algorithm following persons.

In addition to the 'official' work done for this project, there were many unofficial collaborations. The Kinect server, which was written to make the control of the Kinect so much easier, is also a joint effort. Before I started working on the person following I had already setup the core of the Kinect server, but there were many things that needed to change for it to become useful in the Robocup@Home software. These modifications include the filtered RGB/depth data, the RGBXYZ pointcloud, a service call to retrieve a pointcloud and some other minor modifications. These were all implemented and tested together with Aswin and Floris.

Also, OpenTLD was perhaps unsuitable as a person follower, but it proved to work nicely for regular objects standing on a table. Aswin and I were working on getting this to work for a while so that the older detection system from the robot could be easily swapped with OpenTLD.

D Software Improvement Group feedback

This appendix will go into the details of the feedback received from the Software Improvement Group at TU Delft. Halfway through the project I was to send the created code for the project to SIG for evaluation. At the time we were testing several tracking methods so I had sent them all to SIG for evaluation. At the end of the project only the FFOD algorithm remained updated so for the final evaluation this is the code that was sent to SIG. For evaluating the different algorithms and for creating image sequences a Kinect server was written for more controllability over the image feed. This code has also been submitted to SIG. The first feedback received from SIG is the following:

"[Aanbevelingen] De code van het systeem scoort bijna vier sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Complexity en Unit Size.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt. Binnen de extreem lange methodes in dit systeem, zoals bijvoorbeeld de 'KinectServer::run()'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// start streaming RGB' of losstaande if-statements zoals 'if (publishCloud ||publishXYZRGB)' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Door elk van deze functionaliteiten onder te brengen in een aparte methode met een descriptieve naam kan elk van de onderdelen apart getest worden en wordt de overall flow van de methode makkelijker te begrijpen.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de langere methoden ook naar voren als de meest complexe methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen."

As a result of this feedback I refactored parts of the Kinect server to shorten the long methods. There are still some methods that are rather long, but I did not see a way to shorten this nicely. The mentioned run() method for example still needs to be able to make decisions on both RGB and depth images. As there are many outputs possible with the RGB and depth data, this method is bound to become somewhat longer. Eventually it ended up as being a sequence of "do I need to output? Then output it.".

I opted not to add unit tests, as this project was more of a research like project. The code that was written was constantly subject to change, creating unit tests in this case would have slowed progress down considerably. Because of the nature of the project, development processes like test driven development were also impossible to maintain.

E Person tracking on depth using thresholds

FFOD was not the first algorithm that was created in this research. The first attempt was to create and modify a particle filter so that it would use depth data for weighing particles. Because of its complexity however, there was not enough time to finish it entirely before the deadline. For that reason this idea was abandoned and a more simpler solution was attempted. The very first result of this solution was not FFOD, it was an algorithm which simply thresholds the image based on the last known depth.

The initial location of the person is used to determine its average distance from the robot, which the following frame will use as a reference. This next frame has a filter, everything outside a certain threshold from the reference depth is discarded. This should leave only objects remaining in the frame that are within a certain distance from the robot, at which the robot expects to find the person it was tracking. The algorithm then locates the cluster of points that is closest to the last known position of the person, using euclidean distance. The threshold in this case is set to the same value as the overall threshold in the FFOD algorithm, 0.1m, however the width of a person should be added in this case, resulting in a threshold of approximately 0.4m. An example of the output from this algorithm can be seen in Figure 11.

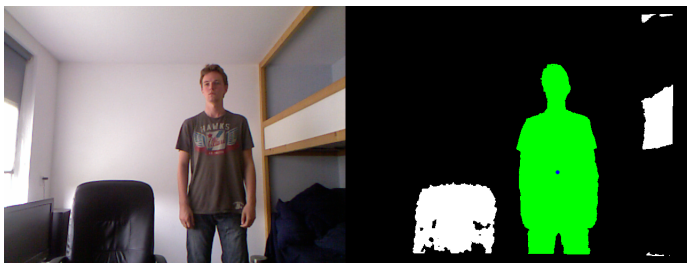

Figure 11: An example of threshold algorithm. This algorithm selects anything within a certain distance from the Kinect and therefore pieces of the surroundings are also segmented, but ignored. The green blob is what the tracker sees as the object.

Below is a description of the steps this algorithm takes:

1. The incoming depth image is filtered on the last known average depth ± the depth tolerance.
2. A mask is created with values 1 for all depth values that still exist after filtering and 0 for those that do not.
3. In this mask, clusters are formed.
4. For each cluster, the euclidean distance is checked from the center of mass to the last known position of the tracked object. The cluster with the smallest distance is kept.
5. If the average distance belonging to this cluster in the depth image is within the threshold depth, then this is considered the latest known position of the target.
6. The center of mass and the average depth are stored for the following frame.

One of the biggest advantages in this algorithm is its simplicity. It is easy to understand, easy to implement and relatively efficient. It is however not hard to imagine scenarios where this algorithm will be flawed. The depth threshold is a hard cut, if the person is extending his arm for example, away from the robot, part of the persons arm will be cut off by this algorithm. Depending on the application this can be a severe issue or not. If the robot is only supposed to follow the average location of a person there is no harm if parts of the person in the frame is cut off, however if the robot has to use this data to monitor gestures and such, it needs to see the entire body. Additionally if someone walks in front of the tracked person within the depth threshold, there is a bigger chance the tracker will 'skip' to this person instead.

Another disadvantage compared to the eventually created FFOD algorithm, is that this processes the entire frame. Although the calculations done are computationally not very expensive, the FFOD algorithm does even less calculations and is therefore more efficient than this algorithm.

References

[1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. Technical report, Carnegie Mellon University, 2002.

[2] Richard W. Bohannon. Comfortable and maximum walking speed of adults aged 20-79 years: reference values and determinants. Technical report, University of Connecticut, 1997.

[3] Floris Gaisser, Maja Rudinac, and Pieter P. Jonker. Robust face recognition and online learning for robotic applications. Technical report, BioRobotics Lab, Delft University of Technology, 2012.

[4] Germán Martín García. Bobot-d benchmark. `http://www.iai.uni-bonn.de/~martin/tracking.html`, July 2012.

[5] Germán Martín García, Dominik A. Klein, Jörg Stückler, Simone Frintrop, and Armin B. Cremers. Adaptive multi-cue 3d tracking of arbitrary objects. Technical report, Carnegie Mellon University, 2012.

[6] D.M. Gavrila and V. Philomin. Realtime object detection for "smart" vehicles. Technical report, DaimlerChrysler Research and University of Maryland, 1999.

[7] Patrick Goebel. Pi face tracker. `http://ros.org/wiki/pi_face_tracker`, July 2012.

[8] Ismail Haritaoglu, David Harwood, and Larry S. Davis. Real-time surveillance of people and their activities. Technical report, 2000.

[9] Michael Isard and Andrew Blake. Conditional density propagation for visual tracking. Technical report, University of Oxford, 1998.

[10] Carl DiSalvo Jodi Forlizzi. Service robots in the domestic environment: A study of the roomba vacuum in the home. Technical report, Carnegie Mellon University, 2006.

[11] Zdenek Kalal, Jiri Matas, and Krystian Mikolajczyk. P-n learning: Bootstrapping binary classifiers by structural constraints. Technical report, University of Surrey and Czech Technical University, 2010.

[12] Dominik A. Klein, Dirk Schulz, Simone Frintrop, and Armin B. Cremers. Adaptive real-time video-tracking for arbitrary objects. Technical report, Rheinische Friedrich-Wilhelms-Universitat Bonn, 2010.

[13] Michael Montemerlo, Sebastian Thrun, and William Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. Technical report, Carnegie Mellon University, 2002.

[14] RobotShop. Hokuyo urg-04lx scanning laser rangefinder. `http://www.robotshop.com/ca/hokuyo-urg-04lx-laser-rangefinder-2.html/`, July 2012.

[15] Peter M. Roth, Helmut Grabner, Danijel Skocaj, Horst Bischof, and Ales Leonardis. On-line conservative learning for person detection. Technical report, Graz University of Technology and University of Ljubljana, 2005.

[16] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B. Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. Technical report, University of Bonn and University of Freiburg and University of Washington, 1999.

[17] TUDelft. Delftrobotics. http://www.delftrobotics.nl/, July 2012.

[18] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. Technical report, Mitsubishi Electric Research Laboratories, 2004.

[19] Wikipedia. Kinect. http://en.wikipedia.org/wiki/Kinect, July 2012.

[20] Wikipedia. Moscow method. http://en.wikipedia.org/wiki/MoSCoW_Method, July 2012.

[21] WillowGarage. Opencv. http://opencv.itseez.com/modules/video/doc/motion_analysis_and_object_tracking.html, July 2012.

[22] Takashi Yoshimi, Manabu Nishiyama, Takafumi Sonoura, Hideichi Nakamoto, Seiji Tokura, Hirokazu Sato, Fumio Ozaki, and Nobuto Matsuhira. Development of a person following robot with vision based target detection. Technical report, Toshiba Corporation, 2006.