Cover page

School of Graduate Studies page

Table of Contents

[1. Introduction 1](#_Toc322545587)

[2. Harlie 2](#_Toc322545588)

[3. Kinect Evaluation 3](#_Toc322545589)

[3.1 Calibration 4](#_Toc322545590)

[3.2 Discrimination Between Users 7](#_Toc322545591)

[3.3 Limited Field of View 8](#_Toc322545592)

[3.4 Moving Base Problem 9](#_Toc322545593)

[4. Pan Mount 13](#_Toc322545594)

[4.1 Performance 15](#_Toc322545595)

[4.2 Summary 17](#_Toc322545596)

[5. Person Tracking 19](#_Toc322545597)

[6. Planning 20](#_Toc322545598)

[6.1 Point-point planner 20](#_Toc322545599)

[6.2 Overall planning 22](#_Toc322545600)

[6.3 Goal Generation 23](#_Toc322545601)

[6.4 Future Work 24](#_Toc322545602)

List of Tables

[Table 1: Conditions for Replanning 22](file:///C:\Users\Bill\Documents\_School\_Spring%202012\thesis\Document\Bill_Kulp_Thesis_2012_04_18.docx#_Toc322545603)

List of Figures

[Figure 1: Kinect's “psi” calibration pose 5](#_Toc322545604)

[Figure 2: Difficulties arise in tracking a user in contact with a chair 7](#_Toc322545605)

[Figure 3: Obstacle avoidance may lead to target loss due to Kinect’s limited field of view 9](#_Toc322545606)

[Figure 4: Tracking performance of Kinect under motion 11](#_Toc322545607)

[Figure 5: DP155 Base Pan (left), Phidgets 1066\_0 Servo Controller (right) 13](#_Toc322545608)

[Figure 6: Output from Phidgets 1066\_0, showing position command and open-loop feedback for position and velocity 14](#_Toc322545609)

[Figure 7: Kinect's effective FOV without (left) and with (right) pan mount 15](#_Toc322545610)

[Figure 8: Performance of pan mount in detecting a stationary face 16](#_Toc322545611)

[Figure 9: Tracking performance of Kinect with pan compensation 17](#_Toc322545612)

[Figure 10: Harlie's motion primitives 21](#_Toc322545613)

Acknowledgements

stuff

Title page

# Introduction

Human tracking is a highly sought after application for robotics. The ability to track a human is fundamental for many tasks that require human interaction. For example, a smart wheelchair could follow a nurse through a hallway, or a pack robot could follow a soldier into combat. Although person tracking comes naturally to us as humans, it is a nontrivial job for a machine, requiring the integration of many unreliable sources of information and the creation a model of the environment from changing conditions. Additionally, a method must be developed to allow the robot to plan to its target under changing conditions as the target moves.

# Harlie

Harlie is a mobile robot built on an electric wheelchair base. Harlie is equipped with a has a server [SPECS] and a SICK LIDAR unit used for obstacle detection and localization.

All software was developed for Ubuntu Linux using the ROS (Robot Operating System) framework provided by Willow Garage. Processing was split between two computers. The main computer was Harlie’s server, running software related to planning, steering, and localization. Kinect and person-tracking software was run on a laptop connected to Harlie via Ethernet. The laptop was a Dell Latitude E6510 laptop with a 2.67GHz Intel Core i5 CPU and 4GB of RAM.

# Evaluation of the Microsoft Kinect

The Microsoft Kinect, released in the year 2010, is a human interface device originally developed for the Xbox to facilitate gestural controls and natural user interaction. When used as a game controller, the Kinect is able to track the positions of multiple users in real time, providing the Xbox with their locations in 3D space as well as the position and orientation of their limbs.

The Kinect has a 640x480 RGB camera as well as a 640x480 IR camera. An infrared projector shines a known dot pattern on the scene, and by computing disparity between the known pattern and what is observed from the IR camera, a depth value can be computed for any given pixel. This gives the Kinect great potential as a 3D sensing system. Its retail price of under $150 prices it far below comparable systems on the market.

PrimeSense, the makers of the Kinect’s software, has released an open-source API called OpenNI (Open Natural Interraction) to allow developers to tap into the Kinect’s functionality. In addition to accessing the depth and RGB camera feeds, OpenNI introduces high-level functionality for tracking users’ skeletons through a library called NiTE. With OpenNI and NiTE, the Kinect is able to seamlessly detect and track multiple human users in its field of view. This is obviously very appealing for the application of person tracking.

For this project, it was necessary to mount the Kinect on Harlie, a moving platform. The Kinect is remarkably proficient at its intended task, although when mounted on Harlie, the Kinect is operating outside of its design parameters. The Kinect has a limited field of view (57 degrees) which is problematic when following users through a real-world environment. Normally, when a new user enters the scene, the user must make a calibration pose before tracking can begin. By default, the Kinect has no means of recognizing a specific user from another, relying on spatial and temporal continuity to tell users apart. Finally, the Kinect has trouble when used from a mobile vantage point, being susceptible to bumps and sudden motions.

Most of these issues could be dealt with by patching the skeleton tracking software. Unfortunately, NiTE is distributed as a closed-source binary and there are few options to probe the library’s inner workings and configurations. Higher-level software workarounds had to be employed to make up for some shortcomings of the Kinect largely due to the closed-source nature of NiTE.

## Calibration

By default, whenever the Kinect detects a new user in its field of view it requires the user to stand in a “psi” calibration pose to acquire an accurate measure of the user's limbs. This calibration step takes several seconds and requires both the target and the camera to be still.



Figure 1: Kinect's distinctive “psi” calibration pose

With the Kinect is on a moving base, occasionally the target will be lost due to relative motion or jolts, as discussed later. Upon target reacquisition, frequently the software will not remember the user and will require recalibration. Recalibration would require both Harlie and the target to come to a halt, which is onerous given the goal of smoothly following the target. Luckily, through somewhat of a hack, OpenNI can be instructed to save the calibration of the first detected user and to apply that saved calibration to all subsequent users.

Skipping the calibration step comes at a cost. The distinctive “psi” pose required for calibration reduces the possibility of the robot following the wrong user, because it is highly unlikely that a bystander would make that pose. Without the calibration step, Harlie no longer has an easy way of telling which user to track. Furthermore, when on a moving base, the Kinect tends to classify some inanimate objects such as chairs as users. These chairs would never pass the calibration step, although without calibration they appear as spurious measurements. This issue was resolved by treating the bodies detected with OpenNI as one input to an overall Kalman filter as discussed in chapter [REFERENCE].

An additional issue with OpenNI, the default behavior of the software is to track the entire human body (head, arms, torso, and legs). Full-body tracking is desirable for the Kinect’s intended application as a game controller, although Harlie's Kinect is mounted in such a way that users’ legs are often obscured (INSERT MECHANICAL DRAWING OF KINECT'S FOV). Luckily, OpenNI can be instructed to ignore users’ legs and just track the target from the waste up. This results in better tracking from Harlie’s point of view, but results in an additional tradeoff. Without the shape cues that legs provide, the tracking software loses an important characteristic that can discriminate people from inanimate objects.



Figure 2: Difficulties arise in tracking a user in contact with a chair

## Discrimination Between Users

A major issue with the Kinect is the lack of built-in facilities for discriminating between different users. While in theory the Kinect has the potential to store color and texture information to recognize individuals, in practice, once OpenNI calibrates on a user, no information is stored other than limb measurements. As a result, if a user exits the scene, there is no guarantee that when the user is re-detected that OpenNI will assign that user the same ID. The same is true if a target is momentarily lost due to a sudden bump or relative motion.

The Kinect relies on continuity between frames to maintain a lock on a target, which is perfectly fine for its intended application as a game controller where players never leave the field of view and the Kinect is stationary so the target lock is rarely broken. However, for applications with a moving base, frequent dropouts must be dealt with. My solution as explained in chapter [REFERENCE] is to use the Kinect as one of several inputs to a Kalman filter that tracks the overall hypothesized location of a person.

## Limited Field of View

The Kinect has a field of view of 57 degrees. While this is sufficient for tracking a target with limited freedom from a fixed vantage point, it shows weaknesses for moving targets. When using the Kinect as the sole source of observation, Harlie must constantly face the user (within ±29 degrees) or lose the target. This puts severe constraints on the ability to maneuver and plan paths while maintaining contact with the target.

Even a task such as following a target down a straight hall can be problematic. If an obstacle appears between the user and the robot, the robot must navigate around the obstacle. As part of the obstacle avoidance, the robot will likely rotate far enough that the user leaves the Kinect's field of view, leading to a target loss. When the robot once again faces the user, it will have to re-acquire the user, leading to delay.



Figure 3: Obstacle avoidance may lead to target loss due to Kinect’s limited field of view

The situation becomes even worse if the user doubles back behind the robot. In tight spaces such as hallways, the user will must come close to Harlie when moving behind it. The Kinect’s depth camera breaks down when targets are closer than 2 feet away. Thus, Harlie’s Kinect has a blind spot for close objects. In a hallway scenario, this can result in Harlie being stuck pointing at close range to a wall within the blind-spot range.

## Moving Base Problem

The Kinect was designed to be placed in front of a television to track users playing a game. Mounting the Kinect on Harlie's moving base poses challenges outside of the Kinect’s design parameters. When the Kinect is still, performance is obviously best. The Kinect can detect users rapidly moving through the scene, and it can easily deal with partial occlusion. The Kinect only loses a lock when a target moves very quickly or exits and reenters the scene. The Kinect can be confused if two users come close together, not being able to tell users apart by means other than their spatial positions.

Although mounted on Harlie, relative velocity must be dealt with. A walking pace for an average human is around 1 m/s. For decent maneuverability, Harlie should be able to navigate curves with a radius of 1m. Thus, by informal calculation, Harlie should be able to handle angular speeds of 1 radian/second.

The Kinect is a complicated system and the tracking software is closed-source, so it is difficult to exactly characterize the system’s performance. However, some metric of performance is necessary. A test was performed in which Harlie was rotated back and forth through 1 radian of angle (slightly less than the Kinect’s FOV) with a sinusoidal velocity profile. The Kinect attempted to track a person standing 2m away, shifting his weight from foot to foot (corresponding to 20cm of motion at 1Hz). If the Kinect performed perfectly, it would maintain a lock on the user 100% of the time. In reality, the Kinect periodically drops the user due to bumps and motion. The performance of the Kinect (the percentage of the time that it was able to maintain a lock on the user) was gathered as a function of maximum angular speed.



Figure 4: Tracking performance of Kinect under motion

The Kinect’s performance degrades as Harlie’s angular velocity increases. When the Kinect loses the target, it usually reacquires the target right away, resulting in a flickering effect as the Kinect tries to maintain a lock. With a peak velocity below 0.5 radians/second, the performance is comparable to the case of standing still. The incidence of flickering increases with speed, as well as the chance that the Kinect will lose a target and not quickly reestablish it. At the maximum tested speed of 1.0 radians/second, the Kinect performs very poorly at tracking, maintaining a lock only around 15% of the time.

In general, the Kinect performs well from a slow-moving base. At low speeds, there is not much difference from the Kinect’s stationary performance. At higher speeds, the Kinect performs more poorly. It is hypothesized that this is due partially to relative motion between the Kinect and the target, and partially due to bumps resulting from Harlie’s dynamics of motion.

# Pan Mount

To alleviate some issues inherent with the Kinect, a rotating mount was built to allow the Kinect to pan and face its target. The Kinect has a limited field of view that is problematic when it is being used from a mobile base, and the pan mount greatly expands the effective field of view. The Kinect is most adept at tracking targets with low relative motion, so the pan mount helps by lowering side-side relative motion between the Kinect and the target.

The chosen mount is a ServoCity DDP155 Base Pan (Figure 5). The DP155 is a low-cost, direct-drive pan mount that incorporates a standard hobby servo. The DP155 has a ball-bearing shaft that makes the pan platform very rigid and reduces axial stresses on the servo. The Hitec HS-485B, a mid-range hobby servo, was selected to power the mount.

 

Figure 5: DP155 Base Pan (left), Phidgets 1066\_0 Servo Controller (right)

To drive the servo, several servo controllers were compared and the 1066\_0 PhidgetAdvancedServo 1-Motor was selected. The Phidgets 1066\_0 enables precise open-loop control of a hobby servo at 30 Hz, obeying programmed constraints on velocity and acceleration. For this project, a maximum velocity of 40 degrees/sec and acceleration of 90 degrees/sec2 was chosen. The device is completely powered by a USB port and provides real-time feedback on current consumption as well as open-loop estimates of position and velocity. Phidgets provides a convenient API with bindings in multiple languages to communicate with the device.



Figure 6: Output from Phidgets 1066\_0, showing position command and open-loop feedback for position and velocity

To maximize field of view, the pan mount was placed on top of Harlie and near the cener. [INSERT DIAGRAM]. This required removal of an aluminum mast that previously blocked the front of the robot and the relocation of some electronics. A mount with both pan and tilt capability was initially considered, although it was determined that the Kinect’s vertical field of view was sufficient so tilt capability was eliminated to cut down on complexity and cost.

The TF (transform) API of ROS was used to represent the time-varying transform between the Kinect and the rest of the robot. The head controller software continuously monitors the last known position of the detected person, and directs the pan mount to move to that angle. The head controller repeatedly receives open-loop feedback from the Phidgets 1066\_0 and publishes a transform incorporating the open-loop feedback.

## Performance

The pan mount clearly alleviates one issue with the Kinect, the limited field of view. Without the pan motion, the Kinect has a limited 57 degree field of view. The pan mount provides 180 degrees of rotation, so the Kinect’s field of view is increased from 57 degrees to an effective 237 degrees.



Figure 7: Kinect's effective FOV without (left) and with (right) pan mount

The performance of the pan mount was also tested under dynamic conditions. A subject stood 1.5m away from Harlie, while the Kinect's RGB data was fed into a Haar cascade face detector at 2Hz. The face detector located the subject’s face in Kinect-relative coordinates, which were transformed to world coordinates to account for the motion of the pan mount. If the pan mount and its associated transformations were working perfectly, the detected face would always be in the same world-relative position, no matter the position or velocity of the pan mount.

As shown in Figure 8, the pan mount performed fairly well. Most measurements were less than 5cm from the expected value (standard deviation = 3.7cm). While an error of 5cm would be troublesome for tasks that require high precision, such as mapping, this error does not pose a problem for person tracking. People are large, distinct objects, and this project could easily tolerate absolute error as high as 50cm of error in positions of reported people.



Figure 8: Performance of pan mount in detecting a stationary face

To update Figure 4, the tracking performance of the Kinect was again tested, this time with motion compensation from the pan mount. Figure 9 includes the new data. Somewhat surprisingly, the pan compensation results in decreased performance under 0.8 radians/second. Because a standard hobby servo was used in the pan mount, its motion is not entirely smooth. It is hypothesized that the pan mount introduces some jitter that makes tracking more difficult. At speeds higher than 0.8 m/s, though, the pan mount resulted in notable improvements to tracking. This increase in performance is due to the reduction in relative motion. The decrease in performance in low speeds is tolerable, made up for by the increase in performance at high speeds.



Figure 9: Tracking performance of Kinect with pan compensation

## Summary

The pan mount greatly improves the tracking capabilities of the Kinect from a mobile base, by quadrupling the effective field of view and compensating for some relative motion. The greatest problem with the current pan mount is its susceptibility to bumps and vibrations. A fairly low angular acceleration had to be programmed into the pan head to prevent jolts. In the future, a higher-grade pan mount with a DC motor and encoder could be explored to provide smoother motion. Additionally, a vibration-isolating mount could be explored to shield the Kinect from vibrations arising from the dynamics of Harlie’s motion.

And I hypothesize that the performance of the pan mount in Figure 9 would increase given a smoother mount, exceeding the graph with no pan compensation in all cases.

# Person Tracking

For reasons discussed in chapter 3. , the Kinect alone is not sufficient to provide reliable person tracking.

Because the Kinect is not reliable enough to be used on its own, a multi-sensor approach was adopted based on the ROS people stack. A primary node maintains a Kalman filter to track the target person. The filter node continuously publishes its state, and subscribes to update messages from observation nodes. The output of the filter node also is passed on to planning as a navigation goal. This architecture makes it easy to integrate multiple sources of observation from various sensors. For this project, three sources of observation were used: a face detector, a leg detector, and a custom body detector based on the Kinect.

## ROS People Stack

The face detector is one of the default nodes included in the People stack. The face detector runs an OpenCV cascade of Haar-like features on the Kinect’s camera feed to detect possible faces. The face detector then filters these matches with depth data from the Kinect, pruning faces based on plausible sizes in 3D.

The leg detector is another node distributed with the People stack. It detects legs using a boosted cascade of features computed from a LIDAR scan [*Using Boosted Features for the Detection of People in 2D Range Data, Efficient People Tracking in Laser Range Data using a Multi-Hypothesis Leg-Tracker with Adaptive Occlusion Probabilities*]

## Kinect Body-Detector Node

The final source of observation is a custom body detector that uses the Kinect to track users within its field of view. A reliability layer was added on top of the Kinect’s skeleton tracking to store persistent information about the user, acting as a fingerprint to aid in identification accuracy.

When the system first starts up, the Kinect must be calibrated before use as described in section [REFERENCE]. At the moment that the calibration is complete, a color snapshot of the user is taken. A 2D histogram of the user’s color information is created in hue-saturation space and normalized. The goal is to create a fingerprint that will be unique to the user. The program maintains an idea of the user’s current histogram, and uses this to weed out non-tracked users.

In the program’s main loop, the body detector receives a list of users from the Kinect [IMAGE]. The program computes a color histogram for each user in view, and tries to make an association with the tracked user. The chosen metric for comparing 2D hue-saturation histograms is correlation. For two histograms and , the correlation will equal 1.0 when the histograms are identical, and will drop down to zero as differences increase.

The hue-saturation histogram was chosen in order represent a user’s color independent of lighting intensity. Still, the user’s calibration over time may change because of changing lighting color or differences in posture. The user’s calibration will also change if the user picks up a new article of clothing, or picks an object. Therefore, a method was included to account for changing appearance of the user.

The hue-saturation histogram can be represented by a matrix . Let the user’s histogram at calibration be , the current idea of the user’s histogram be , and the latest measurement of the user as . Over time, given new measurements of , is allowed to drift away from . This is accomplished through a low-pass filter:

Where is slowly pulled in the direction of . With this method, however, it is possible that will drift too far away and the user will be lost. One scenario involves a user picking up a large object. If the object is picked up slowly, will have a chance to adjust to the new appearance of the user. If the user suddenly drops the large object, his appearance will suddenly change and will no longer be valid. To account for cases such as this, if is not successfully associated with , it is compared to the original calilbration . If is successfully associated with , then is shifted back toward with a second low-pass filter:

# Planning

A major component of this project involved dynamic path replanning. Previous attempts at person tracking at CWRU have used the ROS navigation stack, although these attempts failed due to the limitations of traditional planning methods. While traditional point-point planning is fine for static navigation, such as a tourguide robot moving through a fixed series of poses, point-point planning is not suited for dynamic targets such as people. When tracking a person, a traditional point-point planner would need to replan every time that the person moves. This would require the robot to halt every time that the target moves, resulting in unacceptable stuttering.

This project combined a point-point planner with an intelligent rolling-window approach that successfully addresses these issues.

## Point-point planner

This project’s dynamic replanning is built on top of a point-point planning algorithm. The chosen algorithm used is from the ROS SBPL (search-based planning lattice) package, developed jointly by developed by Maxim Likhachev at the University of Pennsylvania in collaboration with Willow Garage [GET REFERENCE].

The SBPL planner is a search-based, ARA\* planner that operates in 3D (x, y, θ) space. The x-y plane is discretized with 2.5cm square resolution, and angles are discretized with resolution . The planner constructs paths from a pre-defined library of motion primitives that may be chosen to correspond to motions of the robot. A cost can be separately assigned to each motion primitive, for example to prefer wide arcs and straight paths and to penalize backing up. As a result, the SBPL planner produces nice, kinematically feasible paths (Figure 10). Previous work at CWRU involved planning using path segments (lines, arcs, spin-in-place), which were a natural fit for the SBPL planner’s motion primitives. Figure 10 shows motion the primitives customized for Harlie, including forward and reverse line moves and arc moves of two different radii.

At every pose along the path, the robot’s boundary is checked for collision against a 2D obstacle map of 2.5cm resolution. The SBPL planner is fast in normal operation; a typical runtime for planning several meters in a relatively clear setting is 0.1-0.2 seconds. The runtime increases for difficult moves, especially those requiring backward motion or squeezes for tight spaces, although the runtime rarely exceeds 1.5 seconds. Thus, the SBPL planner has the speed necessary for dynamic replanning.



Figure 10: Path produced by SBPL planner in presence of obstacles



Figure 11: Harlie's motion primitives

Modifications for this project included a motion primitive file customized for Harlie. The output of the SBPL planner was converted from a series of points to the CWRU path segment standard. Discretization error relating to the planner’s 2.5cm grid was also corrected.

## Overall planning

This project created a planning algorithm to enable dynamic replanning and the tracking of a moving target without the robot coming to a halt. A rolling window approach splits the robot’s path into two sections, referred to as the committed path and the uncommitted path.

The committed path represents a short-term plan that is passed off to Harlie’s steering, and thus cannot be modified without bringing the robot to a halt. The committed path is nominally 1m long, just enough to keep the robot moving for 1-2 seconds. The uncommitted path represents the robot’s long-term plan to get to the goal, subject to change if the target moves or obstacles appear. The uncommitted path can be changed without penalty, as long as its starting pose is constrained to the end of the committed path.

The planner continuously monitors the committed path, trying to keep its length around 1 meter. If the length of the committed path drops below a threshold, path segments are shifted from uncommitted to committed. If the committed path runs out (the robot is taking a long time planning) the robot simply comes to a halt. Setting the nominal length of the committed path involves a tradeoff. If the committed path is too long, the robot will lose flexibility in planning to the target by committing to a path that might be unsuitable in the future. If the committed path is too short, the robot will run out of path before it is able to replan to the moving goal, causing the robot to come to an early halt.

Table : Conditions for Replanning

Conditions for partial replan

* New goal
* Obstacle in uncommitted path

Conditions for full replan

* Robot currently at rest
* Partial replan fails
* Obstacle in committed path
* Target moves behind robot

When the planner gets a new goal, it attempts to perform a partial replan from the end of the committed path. A partial replan is also triggered if a potential collision is detected along the uncommitted path.

If a partial replan fails, the robot is brought from a halt and a full replan is performed, planning from the halt pose. A full replan is also triggered as an emergency reflex when a potential collision is detected along the committed path. A full replan is also done when the robot is at rest and there is no committed path.

Finally, to improve the performance of planning when the target is near to the robot, a full replan is performed if the target to be tracked moves behind the robot. In this case, it is less painful to bring robot to a complete halt than to follow the previous path.

## Goal Generation

For the purpose of person tracking, special consideration must be given to goal generation. Goals are received from the person tracking module, although these are unsuitable for planning. It would be impolite for the robot to attempt to plan directly to the goal, because that space is occupied by the person being tracked.

This project’s solution was to generate a “constellation” of goals offset by varying angles and distances from the target. The positions were chosen based on experience and simulation, to give Harlie flexibility in planning to the target (Figure 12).



Figure 12: Goal constellation, actual goal in green (grid resolution 1m)

Upon generating the goal constellation, each goal is checked for validity, and goals in collision are removed. To keep planning time reasonable, the first four cleared goals are passed to planning. If a full replan is being performed, all goals are kept. If the robot fails to plan to all four cleared goals, it triggers a full replan.

A special case is when the target is close, approximately less than 1m away. In this case, the robot bypasses planning altogether and simply rotates to face the target.

## Future Work

The robot shows weakness tracking users at close range, especially turning around. An alternate planning algorithm could be employed at short ranges to make the response more fluid.