# Multi-modal Person-Tracking:

## Toward Implicit Communication for Peer-to-Peer Human-Robot Teams in Shared Workspaces

Scott C. Livingston
Advisor: Prof. Lynne E. Parker

***Abstract*—I present here a multi-modal system for person tracking designed for mobile robot platforms equipped with laser range measurement and color camera sensors. Basic person tracking is achieved by filtering "leg" circle features extracted from a laser sensor, considered in combination with the target position (with respect to a particular image) estimated by a vision-based tracking algorithm. This work implements and builds upon previous research successes, and is novel primarily in its combination of the individual elements. The final project strives to be easy to use, being documented and based upon an object-oriented design, and should thus represent a nontrivial contribution toward a larger research project to develop methods for peer-to-peer human-robot teams through implicit communication.**

## I. Introduction

Developing methods for cooperative, peer-to-peer human-robot teams, in which communication is achieved implicitly, has the potential for transformative impact. Though much progress has been made in the domain of cooperation and planning in robot teams, "bridging the gap" to include humans in a shared workspace as peers to the robots (e.g., rather than commanders) remains elusive. A central concern is communication between the humans and robots. Clearly, the human could learn a specialized artificial language for issuing commands to robots on the team, entered using a keyboard and display for each robot. However, of much greater importance and utility is intuitive, implicit communication among team members, whether human or robot. A well-trained human team functions in such a manner; imagine the fluidity of formation building and execution of game strategies by a soccer team in the World Cup.

The overall goal of this research is to develop methods for peer-to-peer human-robot teams using implicit communication and acting in shared workspaces. Specifically, the team needs to interact without any explicit commands or pre-task planning. The key is for the humans and robots to understand the intent and capacity of their collaborators and determine how to optimally interact in a given task. "In shared workspaces," means the team acts in the same physical workspace (in contrast, for example, to tele-operated robotics) and works cooperatively to achieve common goals.

For my senior project, I joined a work in progress by Prof. Lynne E. Parker, Mr. John Hoare, and Mr. Sudarshan Srinivasan to develop new methods within this domain of human-robot interaction. The task chosen for analyzing and demonstrating results is cooperative box-pushing, a well-recognized benchmark for research in robot teams that allows for many variations and levels of difficulty (e.g., varying the number and sizes of boxes, the final locations of boxes, etc.). The goal is to have a human begin to push a box and have one or more robots in the vicinity recognize her intention and provide aid without any explicit communication. The purpose of this simple task is to develop and test models and algorithms from which teams can be trained, progressively increasing the difficulty and eventually extending results to other cooperative team problems.

As achieving the overall aims of this (multi-year) research project in a single semester is infeasible, my central contribution is development of a person tracking system. Successful implementation of a method for tracking a target person is key to human-robot collaboration. The ambition of my software is to be easy to use and ultimately integrate into a human intention and activity recognition system, in addition to other later software components of this research. Indeed, as will be described in this report, the final `PersonTracker` class completely encapsulates a target person tracking method into a single object, whose use is as simple as calling an *Update()* method and retrieving the estimated target position. The purpose of this report is to describe my contributions to the larger research project, beginning with a toolchain and patch to the robot server Player, followed by various C++ classes that implement and encapsulate feature extraction and vision-based tracking components (each operating on a single sensing modality), and finally, ending with a description of the person-tracking software. The report culminates in a detailed tracking example.

## II. Related work

The problem of tracking a known target person arises in a number of domains (e.g., security, personal assistance, and live sports monitoring). Its solution is, in some sense, a prerequisite for any sophisticated human-robot interaction (HRI) system. As expected, methods for person-tracking have been proposed, and I briefly describe two prior works and their relation to my system.

The first is a (primarily graduate) student project completed at the University of New South Wales in 2008, titled "Multi-sensor human tracking" [1]. Their system relies on two sensor modalities: laser and color vision. Pairs of leg features are

extracted from laser scan data based on the observation that human legs tend to appear in short sequences of local minima and maxima of distance data (having the appearance of a brief ripple in the laser scan). This leg detection method was proposed in a prior paper; see the students' final report for references. The vision-based tracking method used in their project is very similar to my "motion-focused color histogram correlation-based tracker;" in fact, I based my design largely on the description given in their report. Their implementation differs from mine primarily in that they included a global motion cancellation scheme, to facilitate ignoring changes in the background due to robot (and hence, camera) motion. I extended their approach by including several routines for sorting and pruning candidate image regions. See the corresponding section of this report for details of my work in this regard. Finally, these two sensing mechanisms were integrated with a world model, about which their final report includes little detail.

Another work toward person tracking with multiple sensors is by Fritsch *et al.* [2]. Their primary contributions are twofold. First, a multi-symbol multi-sensor system for "anchoring" mixed percepts to a collection of abstract symbols (e.g., persons) is proposed, thus providing a means for organizing time-varying sensor streams and tracking the presence of known (abstract) objects. Second, their system is demonstrated on a robot using a combination of color vision-based face detection and laser-based leg detection. Leg features are extracted in a manner similar to that of the previously described project at UNSW; specifically, small segments of relatively similar distances are formed from scan data and labeled as "legs" if several weak attributes are satisfied (e.g., width within some threshold, mean and variance of distances of each measurement in this segment are within certain bounds, etc.). Vision-based face detection in their demonstrated tracking system requires an extended explanation, and thus the reader is instead referred to their paper [2].

As will become evident in the rest of this report, my person-tracking system is similar to these previous methods in the sensing modalities chosen, but differs both in the manner of laser-based "leg" feature extraction and in the vision-based object tracking component.

## III. Video recording and playback in Player/Stage: `vd_dump`, `gencamlog`, and a Player patch

In general, experimental research with robots is difficult and expensive. Accidents happen, and they can be costly and delay further research until new parts are shipped, etc. Especially in the early stages of a project, operating in a simulated environment is invaluable for quickly testing ideas without fear of expensive mishaps and without the numerous maintenance-related time requirements of working with a complex robot platform.

A powerful and widely used tool for robotics research is the Player server (see [3] and [4]). In short, Player separates the hardware specific details of a particular robot platform from the "high-level", and more generally and easily applicable, aspects of robotic sensing and actuation. It operates in a manner similar to device drivers in a desktop operating system, which succeed by providing a common, standard interface to abstract devices (e.g., a keyboard) while hiding the details of a specific design (e.g., a Logitech Super Cool Wireless keyboard). Player is a robot "server" in that it serves requests from clients for read or write access to devices that the server manages, such as a Sick LMS (laser range-finder) or a differential drive system.

The primary simulation environment for Player is "Stage." In short, Stage provides simulated sensor streams to a Player server while tracking movement of the (virtual) robot in a given map. For example, laser range readings may be generated using raytracing given a binary image, with nonzero pixels indicating a wall or barrier (off of which the projected laser can reflect). Further details of Stage are omitted here as my simulations were achieved by replaying logged sensor streams; Stage only acted as basis on which Player could run, despite providing no simulated data.

The existing Player software provides a device for writing to and reading from sensor logs. Though there is good support for reading and writing laser range data to plaintext log files, and while there is even support for recording camera images and timestamps to a log in Player 2.1, there is currently, at the time of this writing, no built-in means for reading back a stream of camera images. "Playing back" a recorded sensor stream may seem trivial at initial consideration, being (one might think) naively achieved by stepping through a log file and sequentially feeding the sensor data to a robot process. However, such an approach does not accurately simulate the timing of arrival of the recorded images. Accurate stream replay is important here because most computer vision algorithms are computationally heavy, by the very nature of the domain, and hence we might reasonably expect some frames to be dropped during otherwise good performance. Allowing the robot to experience all images, perhaps even "pausing time" within the simulation is clearly not representative of an experimental setting.

As a preliminary step to my person-tracking work, I created a toolchain for recording laser and camera sensor streams, generating a new log file by inserting timestamped camera image entries into an existing Player-created logfile, and then replaying the sensor data through Player (attached to Stage) such that robot software can be written to use this sensor stream and, *without need for recompilation*, can also be used experimentally on an actual robot through a Player server running on it. Though this effort is not the centerpiece of my work, it is sufficiently useful and well documented to be mentioned here. A formal software patch may be later submitted to the Player/Stage project.

The toolchain consists of the following:

1) `vd_dump` is a program that coordinates writing a stream of laser range-finder distances through the Player "writelog" device to a log file, while pulling and saving timestamped images from a camera device using a Player camera proxy and OpenCV (see [5]).
2) The `gencamlog` program generates a Player "readlog" format logfile given an existing logfile (presumably

generated by a Player "writelog" device; in this case, it is the laser range data coordinated by `vd_dump`) and provided with a sequence of saved camera images and their corresponding timestamps file. An arbitrary offset time can be chosen when inserting the camera data stream. The new resulting logfile is in plain text and should be bundled with the associated sequence of image files (usually under some directory, such as "camera_samples").

3) I extended the Player "readlog" device to handle camera data entries as generated by `gencamlog` when reading a logfile. The patch has been successfully tested with Player version 2.1.2 and OpenCV 1.0.0. With my extension, a client can access a camera device in Player (e.g., using the C++ CameraProxy object) as usual, i.e., as if it were attached to an online camera.

As with all software written in this thesis, further details, usage examples, etc. are provided in a separate archive, likely called `Livingston-thesis-sw.tar.gz`.

## IV. Feature extraction by inscribed angle variance: the IAVLaserFeatureProxy class

In order to track a person with a laser range finder (called simply a "laser sensor"), features must be extracted that are associated with the target in some way. As the laser sensor is typically positioned about 0.5 meters from the floor (specifically, 30 cm on the robot used in this project), the most natural feature to extract is "legs." The "leg" feature is a semicircle (convex with respect to the robot) with diameters bounded by estimated leg widths being thus labeled. The algorithm used in this project is the inscribed angle variance (IAV) technique [6]. It was chosen both because of its fast performance online and the availability of an existing software implementation.

### A. Overview of method

As its name suggests, the inscribed angle variance (IAV) technique identifies basic geometric features by examining the variance of angles inscribed in candidate segments of a given laser sensor snapshot. The approach is described in detail in [6], but key points are repeated here for completeness.

To understand the IAV technique, consider the sample laser data shown in Fig. 1. The two features of greatest interest are the pair of legs near point (1,1) and the edge of a rectangular box near point (2,0) (perpendicular to the $x$-axis); note that for clarity the walls of the hallway, detected as line segments, are ignored. The basic steps are to

1) Segment the laser scan based on intervals of distances that are relatively close in value; with respect to our example, we would consider 3 segments (corresponding to 2 legs and a side of the box);
2) For a given segment, define an inscribed angle to be the angle formed by connecting a line from each non-end point of the segment to each of the end points; hence, given a segment of $n$ data points, we find $n-2$ inscribed angles;

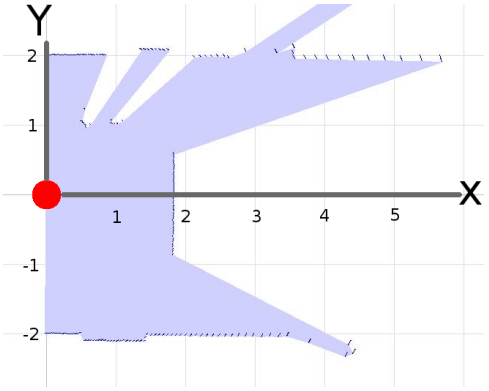


Fig. 1. Sample laser scan from an experimental box-pushing recording, conducted on April 19, 2009. The large red circle on the left indicates the position of the laser range sensor, as mounted on a Pioneer robot. Returned distances are shown in the robot coordinate frame; units are meters. Near (2,0) is a rectangular box, and near (1,1) is a person. The recording took place in a hallway that is coaxial with the $x$-axis.

3) For each segment, calculate the mean and variance of the inscribed angles;
4) Finally, classify the segment as one of several possible feature classes (e.g., line, arc of a circle, "leg") by comparing variances to experimentally determined thresholds (cf. [6]); possibly apply a few other tests, e.g., to determine whether a semicircle opens toward or away from the robot.

In the example scan of Fig. 1, the IAV approach would thus yield a line feature for the box edge and two separate "leg" features (actually, circle features with diameters below a supposed maximum human leg width).

As is clear in the above outline, the IAV technique has only linear complexity, with the heaviest computations being those of the mean and variance for each candidate segment. This aspect of the method is highly attractive considering the complexity of the vision-based trackers described later in this report and the need for online operation of the person tracker.

### B. My implementation

As mentioned earlier, the authors who originally proposed the IAV technique also provided an open source Player driver implementing it, in addition to a simple leg-based "person" feature extraction routine; see [7] for the software, maintained by João Xavier. Unfortunately, at the time of writing, documentation on the software is quite limited, making its use very difficult and time-consuming because it consists of three separate components all of which must be built and orchestrated in a unique manner. Further, once successfully installed, leg features extracted using the software are only available through an special interface, different from typical access to Player-served devices, and necessarily including a 3-D viewer (possibly precluding some applications).

Therefore, I stripped the laser feature extraction code from this collection of software, corrected some minor errors in the source code, created a Makefile for generating a stand-alone library of the result, and documented the key aspects of extracted feature structures. This last item is important because

all features use a common "fiducial" structure while giving different meaning to the fields of this structure depending on the feature type.

Finally, I created a C++ class, called `IAVLaserFeatureProxy`, that inherits from the Player client library class `LaserProxy` and encapsulates the IAV-based feature extraction process. Thus, obtaining features from the current laser scan data is as simple as calling the *Update()* method and then accessing individual features by the *GetFeature()* method. Documentation as well as an example program are provided with the software of this project (see directory `IAV-legtrack`).

## V. KERNEL-BASED OBJECT TRACKING: THE KERNELOBJTRACKER CLASS

Vision-based object tracking is a well-studied and difficult problem and could very well have been the sole focus of my senior project. However, as the task of visually tracking a person is only a small component (that need not be optimal) of the larger research project of implicit communication in peer-to-peer human-robot teams, so I selected two existing vision-based tracking methods to implement: kernel-based object tracking (described in this section), and motion-focused color histogram correlation-based tracking (described in the next section). The final person-tracking system uses a combination of the two.

Kernel-based object tracking, proposed and investigated in [8], is an algorithm for tracking a target object in which a distance metric (in the precise mathematical sense) is defined over the image and from which a gradient is derived for iteratively moving toward local minima that represent probable locations of the target object.

### A. Overview of method

A brief description of the kernel-based object tracking algorithm is provided here for completeness. For details of the algorithm, extensive examples and a mathematical derivation of the distance metric (including proof that it is indeed a metric over the feature space), see [8].

The algorithm must be initialized with a model of the target. This is achieved by providing an ellipse enclosing the region of interest (e.g., a person's face) in a training image of the target. We assume features are colors that map to an $m$-bin color (i.e., hue) histogram, normalized to sum to 1, hence a probability density function of quantized pixel color. The selected ellipse is scaled and translated to be the unit circle, centered at the origin. Given the set of points inside this circle $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, a target model is defined to be the probabilities of features $1, \ldots, m$ (cf. Eqs. (2)-(3) in [8]) by

$$\hat{q}_u = \frac{\sum_{i=1}^{n} k\left(x_i^2 + y_i^2\right) \delta[b(x_i, y_i) - u]}{\sum_{i=1}^{n} k\left(x_i^2 + y_i^2\right)}, \quad \forall u = 1, \ldots, m, \tag{1}$$

where $\delta$ is the Kronecker delta function ($\delta(v) = 1$ if $v = 0$; otherwise, $\delta(v) = 0$), $b(\cdot, \cdot)$ maps the hue (i.e., color) of a pixel to the appropriate histogram bin, and $k : [0, \infty) \to \mathbb{R}$ is a kernel profile, which in this project is of the form (called Epanechnikov profile; cf. Eq. (12) in [8])

$$k(x) = \frac{1}{2} c_d^{-1} (d + 2)(1 - x) \quad \text{for } 0 \le x \le 1. \tag{2}$$

Once the target model is obtained, given a set of target candidate points scaled using the target model (i.e., the $x$ and $y$ scale factors required to achieve the unit circle in the target model) $\{(x_1, y_2), \ldots, (x_l, y_l)\}$, we find the probabilities of features $u = 1, \ldots, m$ by (cf. Eqs. (4)-(5) in [8])

$$\hat{p}_u(x_c, y_c) = \frac{\sum_{i=1}^{l} k\left(\frac{(x_i - x_c)^2 + (y_i - y_c)^2}{h^2}\right) \delta[b(x_i, y_i) - u]}{\sum_{i=1}^{l} k\left(\frac{(x_i - x_c)^2 + (y_i - y_c)^2}{h^2}\right)}, \tag{3}$$

where $(x_c, y_c)$ is the center point of this target candidate, and $h$ is the bandwidth (basically the radius of the target candidate circle in the *scaled* image space).

Intuitively, the kernel profile provides smoothing over the target model and target candidate regions of an image, effectively applying a weight to pixels based on distance to the center of the ellipse.

Given a target model, the distance metric to a target candidate with center $(x_c, y_c)$ is defined to be (cf. Eqs. (6)-(7) of [8])

$$d(x_c, y_c) = \sqrt{1 - \sum_{u=1}^{m} \sqrt{\hat{p}_u(x_c, y_c) \cdot \hat{q}_u}}. \tag{4}$$

The summation inside the radical is a similarity measure (it follows from the above definitions that this measure is between 0 and 1, inclusive) between the target model and a target candidate (e.g., in the most recent camera frame). We may thus derive a gradient in order to maximize this similarity (hence, minimize the distance metric) (see [8], in particular, Sections 4.1 and 4.2), leading to an iterative update rule for the target candidate center

$$(x_c', y_c') := \frac{\sum_{i=1}^{l} (x_i, y_i) \cdot w_i}{\sum_{i=1}^{l} w_i}, \tag{5}$$

where the weight $w_i$ for $i = 1, \ldots, l$ is defined by

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u}{\hat{p}_u(x_c, y_c)}} \delta[b(x_i, y_i) - u]. \tag{6}$$

Once the change in target candidate center between updates is within some small bound (e.g., a single pixel), then the algorithm declares convergence. It is important to note that convergence is guaranteed only to a *local* optimum. The region of the image in which this convergence is guaranteed depends on the initial target candidate center and the bandwidth ($h$ in Eq. (3) above), and is called the "basin of operation."

### B. My implementation

Because no existing, freely available implementation of this kernel-based object tracking algorithm was known at the time of writing, I thus implemented and encapsulated the complete method in a C++ class called `KernelObjTracker`. The software optionally includes adaptive bandwidth scaling,

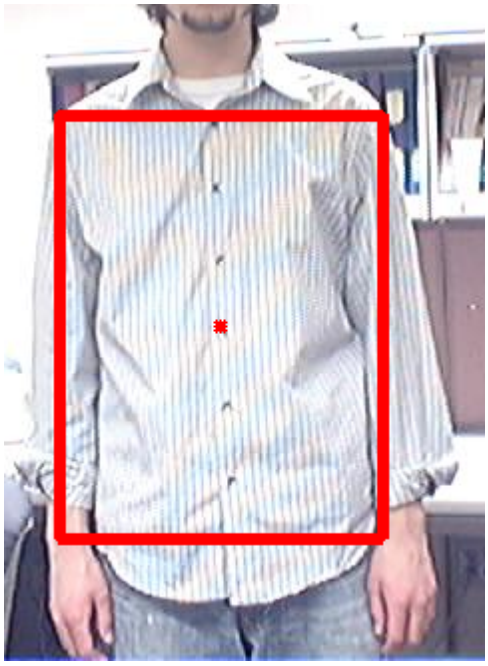

Fig. 2. An example training image used for both vision trackers: the kernel-based object tracking (in `KernelObjTracker`) and the motion-focused color histogram correlation (in `ColorHistTracker`). The trackers use the image region bounded by the red rectangle to initialize their target models. The red dot marks the center of the rectangle. Note that in the kernel-based method, the target model uses points in an ellipse with axis end-points at the mid-points of the rectangle sides, as opposed to all pixels in the bounding rectangle. The rectangle (i.e., the bounding region used for the target model) has size $162 \times 212$ pixels. The fixed kernel bandwidth $h = 0.6$.

though in practice I found automated adaptation to be poor for this application; this may be due to the tracked person frequently entering and leaving camera view, resulting in poor local optima.

The `KernelObjTracker` class assumes a feature space defined over an $m$-bin hue (i.e., color) histogram, where the number of bins $m$ is set to 16 by default (found to yield good performance through trial-and-error) but can be specified by the user. Bandwidth $h$ used in target candidate calculations can be easily adjusted, as well as the initial candidate center point in a given image. The algorithm is derived for *any* $m$-bin feature histogram, and my software is written such that extension to other types (e.g., over RGB color space) can be achieved by straightforward modifications. The kernel profile used in the class is that of Eq. (2), with $c_d = d = 1$.

For software documentation and an example program, see the `vision-trackers` directory in the software bundle of this project.

### C. Example usage

Drawing from the extended example at the end of this report, Fig. 3 depicts the (locally) optimal target candidate region bounded by a red rectangle, shown in the last image obtained from the on-board camera. The target model was initialized using the training image in Fig. 2, which is simply passed to a `KernelObjTracker` object as a method argument, with coordinates specifying the target ellipse within

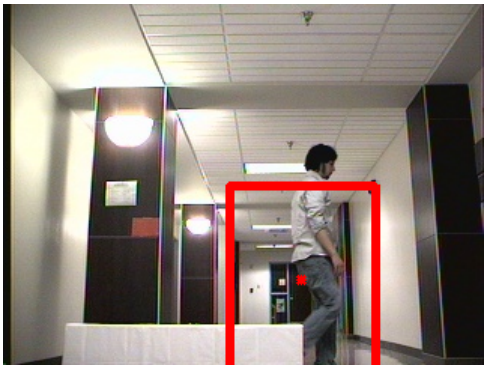

Fig. 3. After building a target model with the training image shown in Fig. 2, shown here is a single frame of kernel-based object tracking from the experimental recording on April 19, 2009. Note that the bright light in the upper-left corner of the image sometimes "distracted" the tracker for periods of several seconds when the target walked under it. The camera frame size is $320 \times 240$ pixels.

the given image. Recall that every completed execution of the algorithm in a particular image is only guaranteed to find a local optimum. In this example, the target candidate sometimes became "stuck," especially when the target walked out of view. For this target model, the light fixture in the upper-left corner of the hallway image (cf. Fig. 3) frequently attracted the tracked region when the target walked out of sight on the left.

## VI. Motion-focused color histogram correlation-based tracking: the ColorHistTracker class

The second vision-based object tracking method implemented in this work is quite simple, lacking the mathematical sophistication of the kernel-based approach described previously. Accordingly, it is computationally lighter (hence, faster online). The basic idea is to find some set of regions of interest in the current image and to select the one which maximizes correlation with the color (i.e., hue channel) histogram of a known target image. My implementation is inspired and driven primarily by the project described by a team of students at the University of New South Wales in their 2008 report [1].

### A. Overview of method

The tracker is initialized by storing the normalized color histogram and the (again, normalized) forward difference of the color histogram of a target training image. The forward difference is a finite derivative approximation of the color histogram and was claimed to yield better performance in previous work (cf. [1]). Given this model of the target and two images (in practice, the previous and most recent frames from the camera sensor), the operation of this tracker is

1) Find regions of motion by
   a) converting given image pair to grayscale,
   b) find the absolute difference between the (now grayscale) images,
   c) apply a $3 \times 3$ Gaussian filter to the difference image to blur it and eliminate some noise,

Fig. 4. After building a target model with the training image shown in Fig. 2, shown here is a single frame of kernel-based object tracking from the experimental recording on April 19, 2009. Correlation is 0.40964 between candidate color histogram derivative (a forward finite difference) with that of the target model. The camera frame size is $320 \times 240$ pixels.

   d) threshold the result,
   e) again apply a $3 \times 3$ Gaussian filter to smooth the thresholded image, and finally
   f) find bounding regions for nonzero clusters of points, which suggest "blobs" of motion;
2) For each region thus found, calculate the statistical correlation of its normalized color (i.e. hue) histogram and the normalized forward difference of said histogram with the target model; and
3) Select one or more regions as likely target locations by sorting candidates based on size, correlation, etc.

In summary, the motion-focused color histogram correlation-based tracker operates in three steps: find candidate regions of interest by determining "blobs" of movement, calculate the correlation of the color histogram (or its derivative) of each region with that of the target model, and finally prune the remaining candidates based on some combination of size and magnitude of correlation.

### B. My implementation

This entire process is encapsulated in a C++ class, `ColorHistTracker`, which provides a simple means for initializing a training image, finding regions of interest and their correlation with the target model given an image, and various sorting and filtering routines to easily select a "good" region (depending on the application). Please see documentation and an example program under the `vision-trackers` directory of the software bundle for this project.

### C. Example usage

Continuing with the experimental recording briefly introduced earlier (and the center of analysis later in this report), an example tracking frame returned by a `ColorHistTracker` object is shown in Fig. 4. As with the kernel-based object tracker, it was initialized with the training image shown in Fig. 2.

The greatest weakness of this vision-based object tracker is the requirement of motion to select candidate regions. In the present application, for consistent tracking it requires the *region* of the target person on which the target model is based to be in continual motion. This is clearly an undesirable property for use as a central tracking mechanism; hence, as described in the next section, this method plays a supporting role for the kernel-based algorithm in the final system.

## VII. Tracking people: the PersonTracker class

The previously described components are now combined into a larger system for tracking a target person, selected *a priori*. I first present the method for tracking and then briefly describe the implementation.

The person tracking system consists of two main sensing modalities: laser range finding and camera vision. The laser process is used simply to extract from scan data "leg" features and short segments that may be a pair of legs (so close together as to no longer be detected as two separate arcs). The vision process uses a fusion of results obtained from a kernel-based object tracker and the motion-focused color histogram correlation method. Before a person can be tracked, the target model must be initialized with some training image for the vision component.

The algorithm for tracking is as follows. Note that the target person position is maintained *in the robot coordinate frame*. We assume a target model has been formed according to the requirements of each vision-based tracker component. Each iteration begins with the raw camera image obtained from the last update, and with the last estimated target person position $(x_{n-1}, y_{n-1})$.

1) Receive most recent sensor data: an image from the camera, and a set $\{(r_i, \theta_i)\}_{i=1,\ldots}$ of laser scan data.
2) Apply IAV-based feature extraction to the laser scan data to obtain a list of "leg" features (circles of sufficiently small radius) and "leg pair" features (cluster of points (not necessarily circular) with width sufficiently small).
3) Apply motion-focused color histogram correlation-based tracker to current camera image and the image from previous iteration.
   a) As specified in this method, candidate regions are first found by bounding "blobs" of motion.
   b) The correlation found for each candidate region is between the forward difference (i.e., approximate derivative) of the color histograms.
   c) The set of candidate regions is pruned to only include regions of area at least 1000 pixels$^2$.
   d) Of the remaining regions, only the region of highest correlation is kept.
4) If no region was returned by the previous step, then go to Step 6. Otherwise, compute the 1-norm (i.e., Manhatten distance) between the center of the region returned by the previous step and the target candidate center found by the kernel-based object tracker in the previous iteration (i.e., the point with which the target candidate would have been initialized for the kernel-based object tracking algorithm applied to the current image).
5) If 1-norm found is greater than some threshold (1000 pixels is used in this work) and if the correlation of

the region is above some threshold (0.8 is used in this work), then initialize the target candidate center of the kernel-based object tracker to the center of this region. Otherwise, leave the initial target candidate center unaltered (i.e., the center found in the previous iteration).

6) Apply the kernel-based object tracking algorithm to the current image, and thus obtain an estimate of the target person location in the current image.
7) Find the shortest (Euclidean) distance between the previous target person position (i.e., the person position from the previous iteration) and the "leg" features extracted from current scan data.
8) If the minimum such distance is below the maximum distance the person could have moved since the last update (i.e., max person speed multiplied by elapsed time, plus a maximum "person diameter"), then update the target person position to be that of the nearest "leg" feature and terminate this iteration. Otherwise, continue to next step.
9) Find the shortest (Euclidean) distance between the previous target person position (i.e., the person position from the previous iteration) and the "leg pair" features extracted from current scan data.
10) If the minimum such distance is below the maximum distance the person could have moved since the last update (i.e., max person speed multiplied by elapsed time, plus a maximum "person diameter"), then update the target person position to be that of the nearest "leg pair" feature and terminate this iteration. Otherwise, continue to next step.
11) Using a pixel-to-radians conversion ratio (experimentally determined): if the current vision-based estimate of the angle to the target person (i.e., $\theta$ in the robot coordinate frame using polar coordinates) differs from the angle to the target person found in the previous iteration by less than some mismatch allowance parameter, then update the target person position to have (in polar coordinates) the radius found in the previous iteration and the angle given by the current vision-based estimate, and terminate this iteration. Otherwise, continue to next step.
12) As all tracking attempts have failed, declare the person position to be unknown.

If the target person is lost, then she is (re)located by

1) selecting the laser-based "leg" feature differing least with the angle found by the current vision-based estimate if this difference is below the permitted mismatch parameter; or
2) selecting the laser-based "leg pair" feature differing least with the angle found by the current vision-based estimate if this difference is below the permitted mismatch parameter.

Otherwise, if no such match between the laser and vision-based estimates is found; then the target person position remains unknown.

This algorithm is implemented in a C++ class called `PersonTracker`, which encapsulates the entire process. An iteration of the person tracking algorithm is performed by simply calling the *Update()* method, whence the target person position and the state of the system (either target location is "known" or "uninitialized") are easily retrievable. The class directly handles all interactions with the camera and laser devices served by Player, given the robot to which they are attached at instantiation. For more software documentation and an example program, see the `PersonTrack` directory in the software bundle of this project.

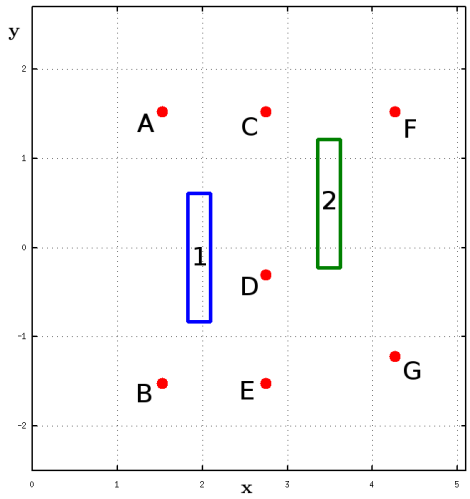


Fig. 5. Experimental setup of laser and camera sensor recording conducted on April 19, 2009. The rectangles labeled 1 and 2 indicate positions of a box respectively at the first and second half of the recording. The red circles are labeled with letters and indicate known locations at which the target person paused briefly during his trajectory (see the text for an explanation). Coordinates are with respect to the robot frame; units are in meters.

## VIII. An Example Experiment

To support the claim that this person-tracking system will be useful toward the ends of the larger research project of peer-to-peer human-robot teams in shared workspaces, an extended box-pushing experiment was conducted.

The goal of the experiment was to fix the robot (with camera and laser range finder attached) at a known location and record a few simple behaviors of a human interacting with a rectangular box that is tall enough to be detected in laser scans. In order to compare positions estimated by my person-tracking system with actual positions of the target person, 7 reference points were marked on the floor (see Fig. 5), and their coordinates with respect to the robot were measured. Further, the beginning and final positions of the box were marked, and the coordinates with respect to the robot were also measured.

During the recording, the target person walks around the box in its initial position (labeled "1" in Fig. 5) several times, then pushes the box (such that the box does not leave contact with the floor) into its final position (labeled "2" in Fig. 5), and finally walks around the box again several times. A total of 3000 camera images were recorded, with a total run time of 231 seconds. Whenever the person walked over or paused at one of the floor reference points, the label for that point

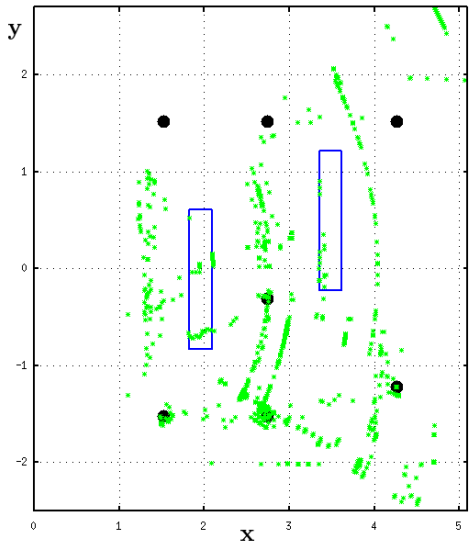


Fig. 6. Plot of all known positions of the target position over the course of his trajectory (green markers). The outlines of both box locations and the floor reference points are repeated from Fig. 5 for clarity. Coordinates are with respect to the robot frame; units are in meters.

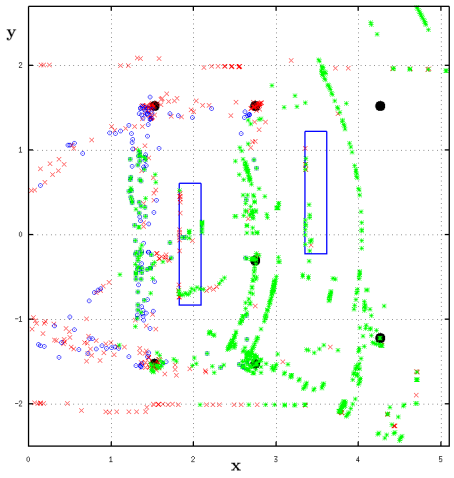


Fig. 7. Scatter plot of all single leg and leg pair features extracted during the experiment (blue and red markers, respectively), and all known positions of the target position over the course of his trajectory (green markers). The outlines of both box locations and the floor reference points are repeated from Fig. 5 for clarity. Coordinates are with respect to the robot frame; units are in meters.

was noted. Thus a string sequence of floor reference labels is formed by the target person's path (cf. Fig. 5 for locations and names of references):

1) B, A, B, A, C, D, E, B, E, D, E, B, A, C, D, C, A, out-of-view
2) A, C, D, C, A, B, out-of-view
3) . . . box is pushed to a new position. . .
4) G, E, B, out-of-view
5) E, D, C, F, G, E, G, F, C, D, E, B, out-of-view
6) B, end-of-recording.

After the sensor streams were recorded, I applied each of the major system components of this senior project to data replayed using the Player/Stage software, with my patch to support playback of the camera image stream. Figs. 6 and 7 show all target person positions found over the course of his trajectory, with box locations and floor reference points included for comparison; IAV laser-based "leg" and "leg pair" features extracted during the recording are also included in Fig. 7.

Finally, a string of visits, defined to be moments of passing over or briefly stopping at floor reference points was generated using the output of the person-tracking system, where a reference point was declared if the target person was found to be within 50 cm of the true measured coordinate. The resulting sequence is

B, E, F, G, A, D, E, B, E, D, E, B, A, D, C, D, C, C, A, F, D, G, G, E, B, E, D, C, F, C, F, G, E, G, F, C, D, E, B

### A. Discussion

The results suggest that the developed person-tracking system is most useful in two respects. First, it allows a target person to be tracked when laser-based leg data ceases to be available, e.g., when the target walks behind a box. The arcs of green points behind both box positions with respect to the sensing robot are periods during which the tracker updated the person's position using only the vision-based estimate, hence the constant radius but changing angle, leading to an arc-like appearance.

Second, the person-tracking system allows a particular leg or pair of legs, as extracted from laser scan data, to be tracked, with preference given to the supposedly most reliable feature: single leg semicircles. With required confirmation from the vision-based component in order to initially locate the target person, while still allowing use of the less distinctive "leg pair" features during active tracking as needed, the overall system seems to perform well, at least as suggested by this initial test.

## IX. Conclusion and Future Work

In this senior project, a multi-modal person-tracking system is proposed and demonstrated in a passively sensed box-pushing experiment. A collection of software has been developed, each component of which can be used alone, and which has been combined to form a person-tracking method, available as a C++ class.

Before practical use in the larger research project of peer-to-peer human-robot teams, or in many other human-robot interaction projects, several enhancements should be made to the proposed person-tracking system. First, the sequence of estimated target person position should be filtered and more intelligently predicted, perhaps with some type of Kalman filter. Second, installation of a PTZ (pan-tilt-zoom) controller for the camera would effectively extend the vision-based tracking to a much wider viewing area, making the region of the coordinate space visible at least commensurate with that of laser scanning.

Once these extensions are made, many more experimental trials should be conducted in order to establish the statistical significance of claims that this person-tracking system performs "well" or "robustly."

## X. Acknowledgments

The author wishes to thank Dr. Lynne Parker for her advising throughout this project and for leading a group of fun and interesting students. Additional thanks are given to graduate students John Hoare, for showing me how to operate the Pioneer robots, and Richard Edwards, for good discussions on research and many pointers to papers relevant to my work.

## References

[1] H. G. Marti, J. Foster, N. Wahi, and T. Rahilly, "Multi-sensor human tracking," 2008, (unpublished) Univ. of New South Wales course project report.

[2] J. Fritsch, M. Kleinehagenbrock, S. Lang, T. Plötz, G. Fink, and G. Sagerer, "Multi-modal anchoring for human-robot interaction," *Robotics and Autonomous Systems*, vol. 43, pp. 133–147, 2003.

[3] B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhatme, and M. J. Matarić, "Most valuable player: A robot device server for distributed control," *Proceedings of the 2001 IEEE/RSJ Int'l Conference on Intelligent Robots and Systems (IROS)*, pp. 1226–1231, Oct.-Nov. 2001.

[4] http://playerstage.sourceforge.net/.

[5] http://opencv.willowgarage.com/.

[6] J. Xavier, M. Pacheco, D. Castro, A. Ruano, and U. Nunes, "Fast line, arc/circle and leg detection from laser scan data in a player driver," *Proceedings of the 2005 IEEE Int'l Conference on Robotics and Automation (ICRA)*, 2005.

[7] http://miarn.cjb.net/.

[8] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, May 2003.