Robotic Video Game Learning Using Self-Detection

Pavel Kazatsker, Ben Cao, Vlad Sukhoy, Alexander Stoytchev

4/22/2011

Contents

List of Figures

1 INTRODUCTION

Trends of modern robotics have moved toward the general interaction and manipulation of objects in the real world. As many people learn when undertaking learning tasks, real-world environments are often too complex to make major strides in advancement of learning. Quite often, a simulated environment is used to diminish noise and allow for the implementation of a system under the terms of the researchers involved. This work will make an effort to perform a simple manipulation in a way that separates the learning and manipulation from the ultimate goal of playing the game. As such, it is possible that this work will provide a significant result even if the

2 RELATED WORK

A number of researchers have perviously focused on a variety of related topics including Object Manipulation, The use of Joysticks, Gameplay, and Self-detection.

2.1 Object Manipulation

A lot of research has been done on the use and manipulation of everyday objects in real and simulated worlds. In the majority of these manipulation tasks, a major subtask in every case is the detection of the object and the understanding of associated affordances [1][3]. Although some researchers treat the actualy task of manipulations as trivial and largely secondary to the detection subtask [2], the majority of manipulation tasks assume that the task interacting with an object is highly complex [1][3].

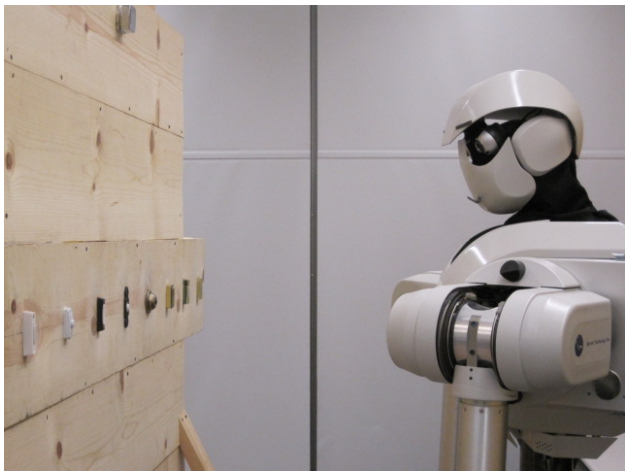
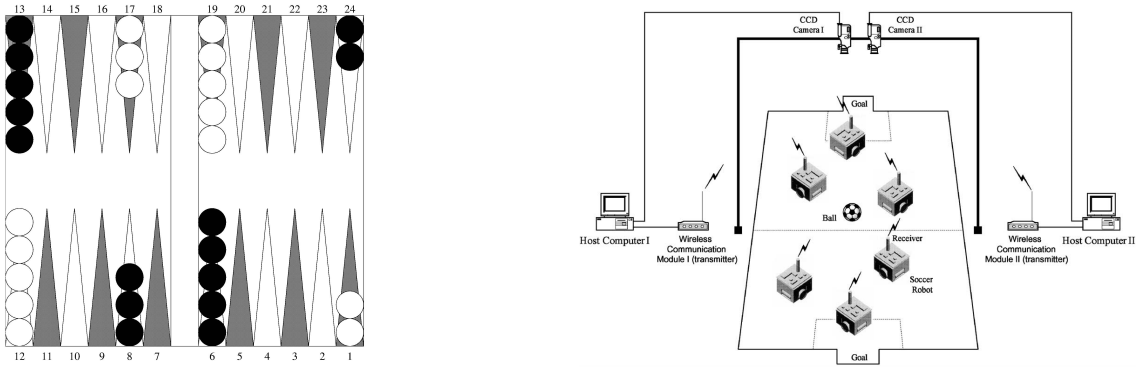


Figure 1: The upper torso humanoid robot at Iowa State University learns the functional components of doorbell buttons by performing a series of eploratory behaviors

A subtask that was intentionally left out of this work is that of grasping the joystick as the first step of the learning process. It should be noted, however, that this is a nontrivial task and can be treated as a rather formidable area of research. A research group at Stanford expends a significant amount of time, money, and energy into the task of learning how to grasp objects for common household tasks [4]. Although this work largely bipases this task, a system with a separate module for grasping of the joystick can be implemented separatly from those discussed here.

2.2 Joysticks in Robotics

There is significant prescedent for the use of Joysticks in Robotics applications, but those applications are largely limited to those in which the robot is controlled by a human operator with the use of a joystick. Such systems have been used to aide disabled individuals in everday activities [5]. Similar joystick-controlled robots are used in more intensive medical applications including ultrasound [6] and Surgery [7]. These Machines have recently been used to safely augment human capabilities in medical applications by improving precision, enabling remote use, and creating immersive simulations for use in the education of new procedures and doctors.



(a) backgammon (b) soccer

Figure 2: Games have always been a research area of interest in AI and in robotics. These games vary vastly in play type and style. Games like Chess, Checkers, and Backgammon are often single-agent games that are often tested against human oponnents. Robot soccer has also become an area of competition for multiple teams of multi-agent robots.

2.3 Gameplay

Playing games has always been one of the classic tasks in artficial Intelligence. The task of effectively playing a game of chess at a high level was one of the classic tasks assigned to artificial agents as far back as the 1950's [8]. In more recent years, artificial agents have reached a level that enables them to compete and defeat human oponnents in a number of games including Chess [9], Backgammon [10] , Poker [11], as well as a plethora of other games from a number of cultures around the world. The task has grown to the extent that a generalized platform has developed to encapsulate novel games as general learning tasks [12].

The games above vary in determinism, completeness of information, and general platform but have the common theme of being turn-based and simulated. There is a notable presence of real-world and real-time games that have a major presence in modern robotics. Most notably, A variety of Soccer playing robotic platforms and contests have been in the robotics scene for years. Much like the turn-based games, game strategy is a sizable research task for these soccer playing agents. A major difference arises, however, with the consideration of real-time elements [13]. Furthermore, the real-world adds a number of physical considersations that were no present before. These considerations include obstacle avoidance and physical robotics controls[14].

2.4 Self-detection

The process of detecting the self has proven essential to a variety of manipulative tasks in biological entities. Studies with Young humans and Apes has shown a strong correlation between the ability to recognize the self in the visual field such as mirrors [15]. With that correlation in mind, the creation of these self-concepts have been a topic of much theory. Possible algorithms for creating this self-concept are often discussed and vary in complexity [16].

Some of these algorithms were, naturally, later implemented in robotic platforms. A platform was developed that explicetly labeled the "self" in the visual field based on the contingency of movement in the visual field to the activation and deactivation of the robot's actuators with some considered delays [17]. Later work fromalized a system for the extraction of the aforementioned delay and reapplied it into the system for a system of self-detection that allowed for improved varification [18].

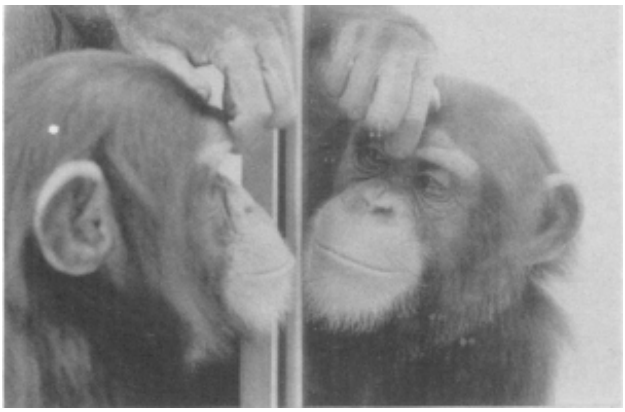
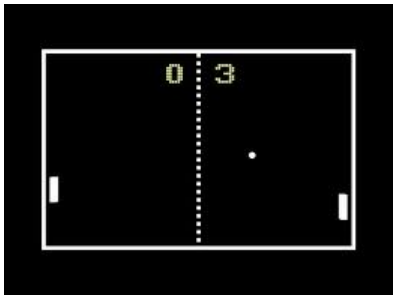


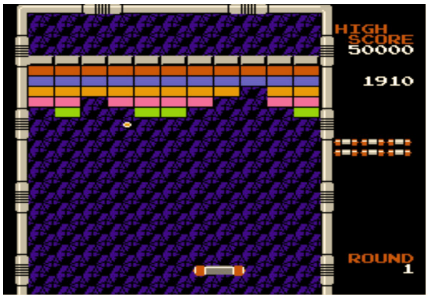
Figure 3: When placed in fronto of a mirror, certain apes will indentify themselves based on the contingency of their movements and the image in the mirror

3 Experimental Setup

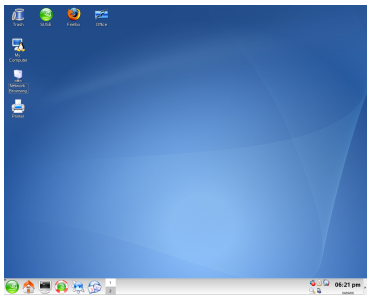
3.1 Games



(a) Pong



(b) Arkanoid



(c) Desktop Simluator

Figure 4: The games used were inspired by a sequence of real-world analogues. Pong and Arkanoid are credited as being two of the earliest games released for play in the home. These games were selected for their simplicity in both design and gameplay. Limitations in this system rendered the original games impossible to use so the games had to be redesigned with graphics and controls that were more suited to the system.

Games were selected for a combination of functionality and historical perspective. The games also had to be modified from their more familiar form to match some of the motion and vision assumptions that are inherent in this system. All together, the three games were meant to demonstrate the capabilities and limitations of this system in the context of easily recognizable games.

The first game was selected for largely historical reasons. Pong was first introduced as an arcade game in 1972. It featured a small ball bouncing back and forth between two paddles controlled by players. The home version of Pong featured a controller with two knobs allowing players to control each of the two paddles on the screen. The game was meant to be a simulation of a head-to-head game of Table Tennis. Later adaptations were made to operate with different controls. Commercially available joysticks were used to replace the dials from the original game.

Pong eventually gave way to a similar game known as Breakout. This game again featured a ball bouncing off of a player-controlled paddle. This time, however, the objective was to hit a number of blocks positioned opposite the player. Breakout also features one-dimensional motion and simple

controls. Later versions of breakout featured a complex patterns, complicated scoring schemes, multiple levels, and unique powerupts. For the sake of this research, the game was kept simple.

The third game selected has no historical context and was created with the intention of demonstrating that this system isn't limited to control in one dimension. This game was a desktop simulator and featured the joystick as a controller for the mouse cursor analogue. The setup of the game is very simple, the player controls a sprite composed of simple gemoetric shapes and the objective is to move that sprite to an object that moves independently of the player's control.

3.2 Joystick and Television

The Joystick and Television are commercially avaiable models that were selected solely for their physical properties. The television was required to be large enogh and have a sufficient resolution for the robot's webcams to distinguish the details of the games. Similarly, the joystick was required to be of the proper dimensions for the robot's hands to grip the controller and manipulate it with relative ease. The selected model of joystick was a Microsoft Sidewinder. A number of different makes and models were used for the Television.

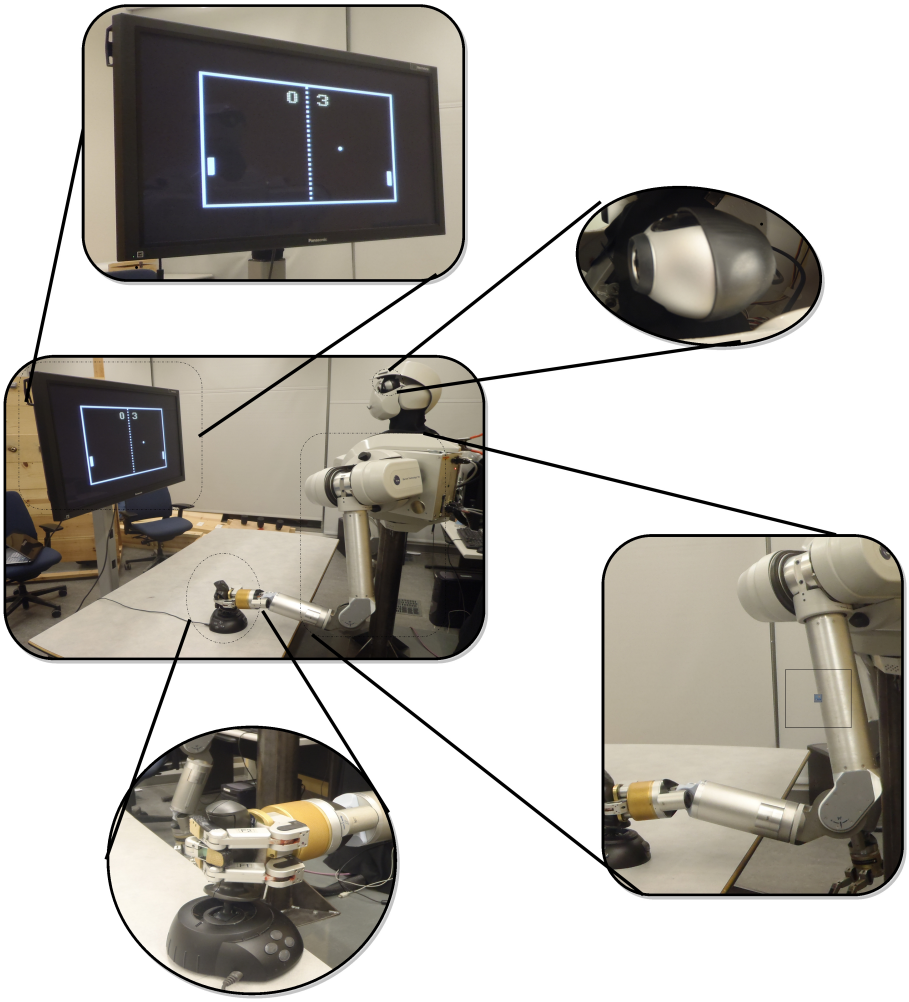


Figure 5: The robot setup as shown here. The robot is equipped with two Barrett Whole Arm Manipulators (bottom right) and two commercially available webcams (top right). Only the left arm and webcam will be used here. Before any computation or babbling is performed, the robot grasps that joystickbottom based on scripted behavior. During the actual running of the system, the joystick will have to be clamped down so that it can be operated properly by the robot. The games will be displayed on a large screen TV mounted on a portable mount (top).

3.3 Robot

The robot used for this experiment is an upper torso humanoid robot. Notable features for this robot include two Barrett Whole Arm Manipulators (WAMs) with Barrett hands, two head-mounted commercial webcams, a head-mounted sensor, and a vibro-tactile sensor placed on one of the fingers. A number of these components and modalities will not be used. For this learning process, only the left arm and left webcam will be used. Audio, proproceptive, and vibro-tactile data will be discarded. With the exception of the hard-coded scripts required for the pre-exploratory behaviors, the learning process will be completely invariant to which arm and which eye is used.

4 Methodology

The methods used to implement gameplay can be broken up into four sections plus the evaluation of the system's performance. The first section describes a system of random movements similar to that performed by infants during their developmental stages. The second is a vision pipeline which describes the way that raw visual input is used to implicetly construct a model representing relevant information used in gameplay. The visual information is piped directly into the third module, Self-Other Separation, which assigns a label to each extracted element to determine whether or not its directly under the robot's control. Finally, All this information is piped into a simple gameplay module which is later evaluated to gain a sense of the system's performance.

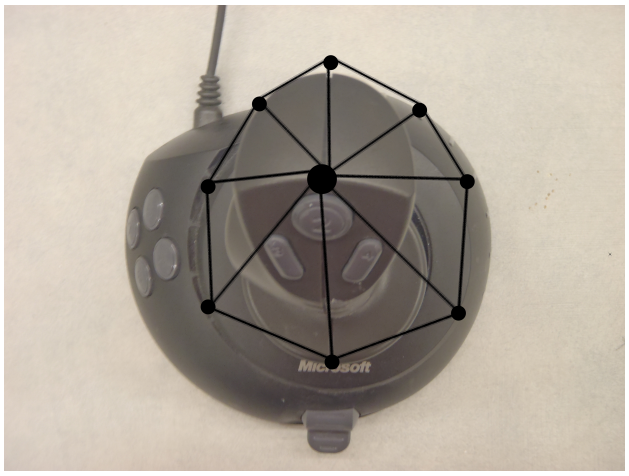


Figure 6: Points are sampled around the functional area of the joystick by selecting a random point within a triange that is formed by taking two consecutive points along the outer polygon displayed in this image and the center point. This region is assumed to be convex in $\Re^7$.

4.1 Pre-exploration

At the beginning of the learning step, the robot and associated system already has certain information. The system starts up with a script that includes the exact sequence of motor commands required for the robot to grasp the joystick. This always has to be done with a specific sequence of commands as the robot must avoid various obstacles en route to the joystick. This aspect of the learning process is both significant and non-trivial but are beyond the scope of this project.

Along with the sequence of motor commands required to grasp the joystick, the robot was given joint locations that approximately correspond to the functional limits of the joystick. Those functional limits are somewhat honed by the robot before the main learning component of the project can take place. To obtain precise functional limits, the robot moves in the direction of the approximated limits as far as it can within the specified torque limits. When the torque limits are reached, the new location is marked as the functional limit of the joystick. There had to be at least 4 such points and they had to be gathered in a clockwise order.

In order for the babbling the joystick to be used properly, a number of joints were left loose. This created some instances where the robot would "get stuck" in one of the limit locations and would be unable to sample the rest. Most of the time this happened, it did not effect game performance as there was enough of a functional area to learn all relevant locations.

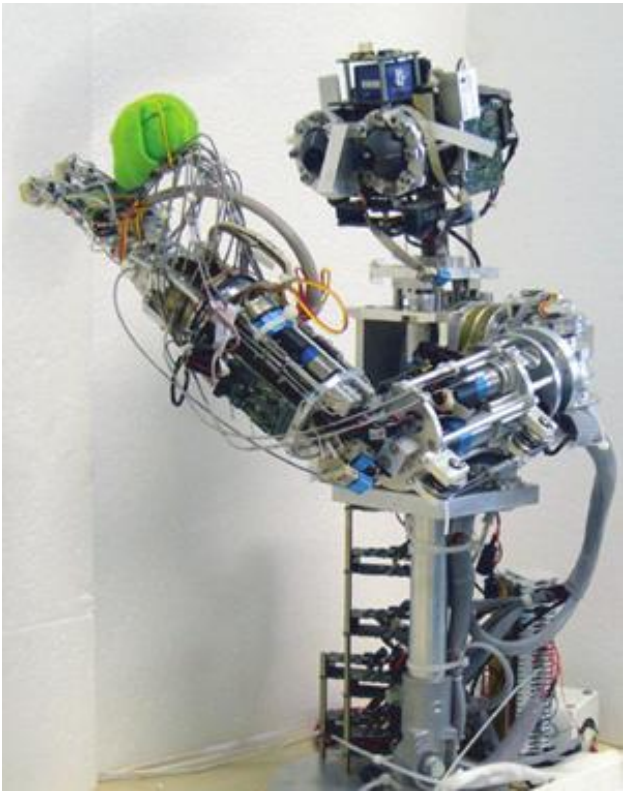


Figure 7: James, an upper torso robot from the Italian Institute of Technology uses babblign for sensory-motor learning [19]

4.2 Motor Babbling

As is the case in many explotarory behaviors in psychology, this system will have a vague notion of a goal without any pre-existing representation of the system in which it exists. This representation is created via a series of random exploratory behaviors. In this case, the robot need only explore a number of postions within the functional area of the joystick (locations in joint space it can move with the joystick in hand without exerting too much torque). Each of the explored positions was stored along with the perceived location of the game components for later recall. Due to the loosened joints and the torque limits, position recall might be imperfect and positions were updated during the later stages.

The positions of the paddle needed to be stored in real time for as many of the babbling steps as possible. Preliminary experimental results revealed that the self-detection could not operate on top of this with sufficient fidelity to differentiate paddle components and store their postions. That is, the confusion created by trying to store the position of the paddle while differentiating the paddle from the ball made it impossible to properly store positions. An assumption was, therefore, introduced that stated that during this babbling phase, the only moving component on the screen would be the one that should be categorized as "self" (the paddle).

4.3 Vision Pipeline

The most computationally intensive part of the system is the system used to parse raw visual information into a representation that is meaningful for the other modules in the system. Ideally, the output of this module is some kind of data structure that would label each component in the game to be accounted for separately and uniquely. This required a number of functional assumptions to

be imposed on the game to manage the process. Each of these assumptions will be outlined as they become relevant.

4.3.1 Reading an Image

Image processing begins with the webcam at taking in images at 15 frames per second at a resolution of 640X480. Excessively complex bacgrounds that take up too much of the image are likely to create too much computational difficutly. It is, therefore assume that most of the image real estate will be taken up by the television and that most of the television screen will be taken up by the game. The physical components of the system are placed in such a way that enforces these assumptions. The television is placed as close to the robot as possible without the robot's arm reaching the television (in order to avoid the unforunate possibility of the robot punching the television in the event of an error). This restriction isn't too unnatural as the television also has to be located on the far side of a table on which the joystick sits.

4.3.2 Feature Extraction

The second step of the Image processing is the extraction of corner features based on the individual images. This is currently done using the OpenCV function *cvGoodFeaturesToTrack*. The nature of this feature tracking function creates the necessity for another assumption. The relevant components of the game had to be textured in order for the feature tracker to identify a sufficient number of points in the image. This becomes necessary due to the original intent of the OpenCV functionality. Things in a simulated environment (in this case the games) tend to be perfectly straight lines. This feature detector, on the other hand, is designed to work for applications using real-world images. Images of physical objects tend to have more corners that make for convinient tracking. Textures were, therefore, added to the games in order to suppliment the number of unique features extracted from a given image.

4.3.3 Optical Flow

Features are tracked between images using Optical Flow functionality. The chosen implementation for this process is the OpenCV and the function *cvCalcOpticalFlowPyrLK* Optical flow outputs with a vector for each tracked feature. Each such vector represents the change in position for a feature between consecutive frames. This process creates another hidden assumption into the assumption. As results quickly proved, using this functionality on images displayed on a television screen is strongly effected by glare. Furthermore, reflection on darker colors create an inconsistency that renders the OpenCV function entirely ineffective. It was, therefore, decided that the backgrounds of all the games must have a white background.

4.3.4 Feature Grouping

Features are then grouped together based on their movement between frames. This process was implemented from scratch using OpenCV and C++ standard library data structures and outputs components which are the atomic unit used for all computations outside this vision framework. The algorithm used for the creation of these components requires the consideration of three consecutive frame images I_1, I_2, I_3. Some information is maintained between frames so assume that these three frames are located at the beginning of the stream and that no prior information exists. The first part of the process is feature detection on I_1 (described earlier). Feature detection produces a set of features F_1. Optical flow is then performed using F_1 and I_2 producing a new set of feature F_2' representing the image of F_1 on I_2. That is, F_2' is the locations of the features found in I_1 in their new locations in I_2. A graph $G = (V, E)$ is then created as follows. Let V be the set of Features in F_2'. Note that

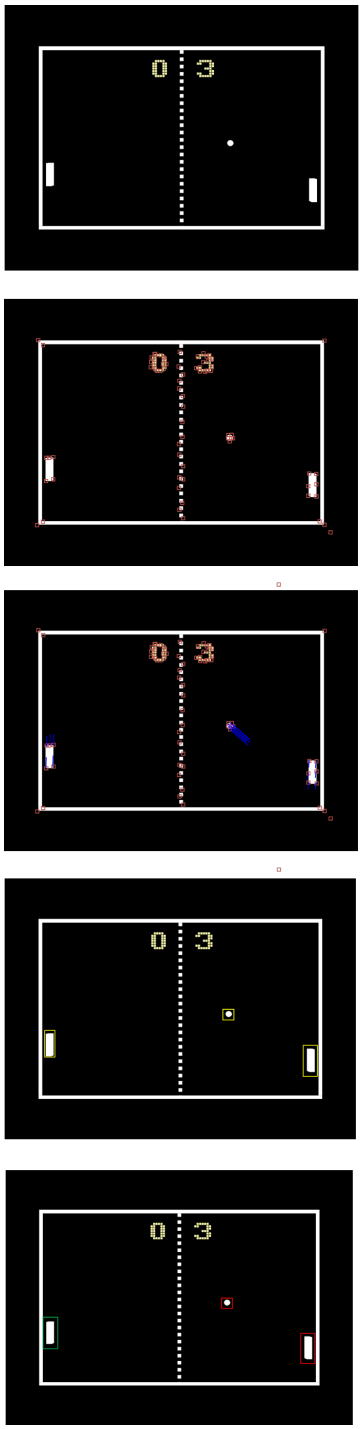


Figure 8: This diagram shows the progress associated the image pipeline. The first image represents the raw imagine taken in by the webcam's equipped in the robot's hand. In practice, the images aren't neatly cropped and contain a visually noisy, background. The second image contains the corner features, highlighted with small red boxes. Optical flow is then performed, which provides a movement vector for each feature. These vectors are represented with blue lines. The features are then grouped together if they have parallel feature vectors. larger such groupings are saved for future computations as image components (pictured in the fourth image as yellow boudning boxes. Each of these components is then subject self-other separation. The last image demonstrates the result of this separation by coloring "self" components in green and "other" components in red.

$|V|$ is not necessarily equal to $|F_1|$ as some of the features are not tracked between frames and are discraded during the optical flow process. The edges E are, then, the the set of (u', v') where $u', v' \epsilon F_2'$, $u, v \epsilon F_1$ are the back projections of u' and v', and $\|u' - v'\| = \|u - v\|$ within some tolerance. In other words, feature pairs that maintain the same distance relative to each other in consecutive frames are connected by an edge in G.

The next step in the process is grouping the the vertices in G into groups that are most strongly connected. The overarching idea is that groups of features that maintained a constant distance from each other compose larger rigid components that can be treated as atomic units. There are two major assumptions to note relating to this principle. The first is that movement is almost entirely two-dimensional. features from objects that move parallel to the to the field of vision will, naturally, shrink and growing thereby decreasing and increasing the distance between features. The second assumption is, therefore, that all movement will be along a plane orthogal to the field of vision. Moving in three dimensions will distort the perceived dimensions of any object and simply break this system.

Formally, the goal of the grouping process is as follows, Find a set of N Components $\mathrm{C} = C_i where C_i \subset V for i = 1, ..., N$ and each Component $C_i = (V_i', E_i')$ is a subgraph in which each $v' \epsilon V_i'$ is of degree at least $|V_i'|/\beta$ where β is an empirically derived connection strength parameter. Its clearly possible for multiple such groupings to exist so the algorithms should prioritize groupings with components that are as large as possible. Without delving too deeply into complexity theory, it can be demonstrated that this problem is NP-Hard. As the number of of features tracked can potentially be quite large (in the order of a few hundred at each given frame), no perfect solution for this problem exists. The algorithm used was, instead, an approximation algorithm that took advantage of the likely geometry of the graph. That is, consider an ideal case in which all the features that form a game element (the paddle or the ball) move together. The graph would, therefore, be composed of two disjoint and disconnected cliques of somewhat different sizes. The number of elements is not an assumption of the system so there may be multiple such cliques. In the ideal case, sorting the vertices by their degree would essentially group each vertex with it's respective component. Iteratively building graphs by testing each vertex by decreasing degree and checking for the connectivity constraint ultimately creates the desired grouping. Naturally, the graph itself is subject to noise introduced at its creation and the game elements won't form perfect cliques and may be connected. This noise should, ultimately, be overshadowed by the connection strength of the features extracted from the game elements.

These components form a running list that is maintained indefinetly (as long as individual components exist). For ease of referece, each component is given a unique ID number. The longevity of each these components is a telling measure of the performance of this tracking system and the performance of Self-detection algorithm and the gameplay itself is highly dependent on how long these components stick around.

4.3.5 validation

After all the grouping is completed, I_1 is discarded and computation begins relating to I_3. The computation required to maintain these components through the following frames is very similar to that of creating the components in the first place. The individual point features of all the components can be treated as though they were just extracted using the feature detection algorithm. This approach has two major downsides. First, it discards the previous grouping making component longevity limited to a single frame. Second, it requires the recomputation of the grouping algorithm described above instead of taking advantage of the known grouping. An additional step is, therefore, added to subsequent frames. Before the feature detector is used on I_3, Optical flow is performed between I_2 and I_3 on all the features from the running component list. Each component is then, validated. That is, a graph of all the components if built and the features that are no longer part of their respective cliques are removed. This process is sufficient to maintain component identities over frames. New Features (those

gathered by the feature detector on I_2 and projected onto I_3) are then superimposed on the graph used for validation. That is, for each new feature, if it connects to $|C_i|/\beta$ vertices of any component C_i, add that feature to the component. All unmatched components then undergo the grouping process described previously. Note that features that the optical flow on the running component features might produce the same individual points as the feature detector on the new image. There is, therefore, the possibility that the same feature can be included twice in the same component. This can be problematic as it can overstate the size of a component (in features) and become computationally intensive. Features that were on the same pixel were, therefore, removed and components were limited to only a few dozen features.

An implicit assumption in the visual model is that only elements that are moving need be considered for any calculation. The excessive amount of stationary point features would make this algorithm very computationally excessive while providing no meaningful results (all the stationary points would inevitable be grouped together all the time). Note, however, that elements only needed to move some of the time in order to still be considered valid.

4.4 Self-Other Separation

As previously stated, the ability to discriminate the self from others in a sequence of precepts is essential for the performance of most manipulation tasks. This principal was, therefore, a major consideration in this work.

The aforementioned separation was performed, as before, by the perception of change temporally relative to the commands sent to the robot's actuators. Unlike previous works, however, it became necessary to consider temporal events in a more continuous matter. For this reason, events were considered with the greatest degree of granularity possible. That is, Each visual frame was treated as a separate event regardless of the onset of visual change or change in the state of the actuators.

An essential assupmtion in the model used here is that a subject would have sufficient dexterity and experience to identify the onset and the halt of the movement its actuators. This assumption was simplified by adjusting all relavant game mechanics so that the position of the player sprite is relative to the position of the joystick. This is done in contrast to many game controls in which the velocity, and not the position, of the player sprite is relative to the position of the joystick. This leads to the following assumption: Movement of the player sprite should correspond with a relatively high degree of correlation to the movement of the actuators.

The ultimate goal of self-other separation should be to identify objects directly under the robot's control (the player sprite) as "self" and everything not directly the robot's control (everything else) as "other". An adaptation of a previous approach was employed for this purpose. Two measures previously existed: Necessity and Sufficiency. Necessity was defined as the fraction of frames in which movement is expected (commands to the actuators) in which this movement is perceived. Sufficiency is the fraction of frames in which movement is perceived in which movement is also expected. Intuitively,

$$Necessity = \frac{ContingentActiveFrames}{ActiveMotorFrames} \quad Sufficiency = \frac{ContingentActiveFrames}{PerceivedMovementFrames}$$

$$Necessity^{-1} = \frac{ContingentInactiveFrames}{ActiveMotorFrames} \quad Sufficiency^{-1} = \frac{ContingentInactiveFrames}{PerceivedStationaryFrames}$$

Figure 9: Four equations were considered when deciding on a sufficient but simple self-detection algorithm. Two of the fractional values were deemed inconsistent or meaningless in the context of this work but the other two were thresholded and combined to create the self-detection algorithm.

if these fractions are high, an object is more likely to be the self. Events in which movement is both expected and found are said to be contingent and the count of these contingent events were the primary measure used for self-other separation.

A number of circumstances rendered these measures both inconsistent and insufficient for self-other separation in the case of these games. Consider an object that is always moving as is the case for the target ball in many of these games. An object that is always moving also happens to be moving when the movement is expected. This situation would would inflate the sufficiency measure rendering it largely meaningless, at least for those objects. Consider also the imperfections created by the intentional lack of pre-existing knowledge of the physical knowledge of the game controller. That is, The robot is likely to make a large amount of movements that have no perceptable effect on the player sprite in the game. For example, vertical movement of the controller in the breakout game would produce no change to the position of the player sprite. These imperfections would deflate the value of the necessity measure.

To correct these issues, two additional measures were considered. These measure were inverse of the pre-existing ones. The inverse necessity measure is defined as the fraction of expected idle events (events in which the actuators are idle) in which no movement is perceived. Likewise, the inverse sufficiency measure is defined as the fraction of events in which no movement is perceived in which none is expected. A quick analysis shows that these two measures by themselves are subject to the same problems presented before. That is, The inverse necessity measure is subject to inflation in the case of often-idle objects and the inverse sufficiency measure is subject to deflation by motor commands that don't produce a movement.

In order to account for the shortcomings of the different measures, the combination of the sufficiency measure and the inverse necessity measure were used. These two measures were seen as making up for each others' shortcomings. Consider the aforementioned problem identified of constantly moving objects that broke the sufficiency measure. Such an object would be in motion even in the absence of action by the motor. This would have the effect of lowering the inverse necessity measure and would cause the object to be correctly classified as "other". Similarly, an often idle object that inflates the inverse necessity measure would deflate the sufficiency measure and such an object would be properly identified as "other."

4.5 Gameplay

The implementation of gameplay was kept simple as obvious algorithms are sufficient to effectively play these games. For all games, the tactic was to store locations that the root reached through babbling and to simply recall positions that would best reach the target location. The target location in all cases is, of course, that of the ball so the goal of the robot is simply to do its best to place itself, or the object it can control, on top of the other object. The robot is clearly unable to actually reach the target in two of the three games but, since the movement is confined entirely to a single line, the closest possible location along the axis of movement is the ideal location for the paddle to be. For this reason, this simple algorithm was deemed sufficient to effectively play these games.

5 Measures of Success

Each of the individual modules is subject to its own evaluation. A separate set of matrix can reasonably be produced for each of the games, the visual model, the self-detection, and the learning process as a whole.

5.1 Evaluating the Visual Model

There is a number of artifacts produced by this type of visual model that is likely to have an adverse effect on the performance of the system. The ability of the system to diminish the occurance of these artifacts is a relevant measure of its success. The first and most prevelant of such artifacts is the tendency for components to appear and disappear. Since these components are the atomic unit for the rest of the system, losing all measures associated with this system can signifcantly hinder performance. The longevity of individual components can, therefore, be a very telling metric. Another correlated artifact is the appearance of components for a small amount of frames (less than 10). A simple count of the number of such components per a unit time can prove a valid indicator of the amount of noise this system introduces. Finally, a measure of precision can be implemented.

5.2 Evaluating Gameplay Performance

The games are direct in their objectives and the objectives themselves can be used as performance metrics. In each of Pong and Breakout, the objective is simply not to allow the ball to get passed the (controlled) paddle. The consistency with which the robot can do this is a valid measure of the performance. Since Pong is a head-to-head game, another potential metric is the objective performance of a system relative to that of a human being. Realistically, however, it doesn't seem likely that the robot could perform well enough to so much as score a point against a human opponent. The metrics for pong will, therefore, have to be constrained to those that can be measured without a human player. Since the desktop simulator has no losing scenario, the performance for this measure can be calculated based on the rate with which the objective location is reached.

5.3 Evaluating Self-detection

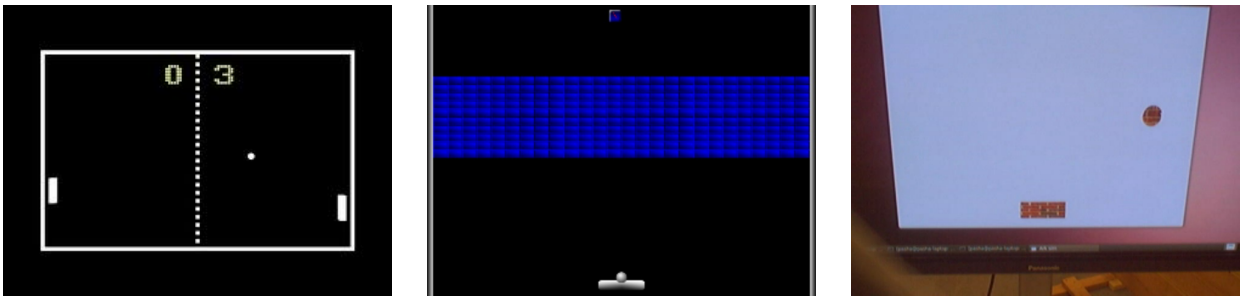
As mentioned previously, the various modules of the system create the necessity for the self-detection algorithm to not only be highly accurate, but for it to converge reasonably quickly. In the current model, a component that disappears then reappears will be assigned a new ID and all calculations for the component are reset. The performance of the system then becomes dependent on the system's ability to properly classify components as "self" or "other" as quickly as possible.

6 Project Progression

This project is naturally prone to a modular design and is likely too large to undertake in one step with a number of milestones. It, therefore, made senes to iteratively improve each of the individual modules in such a way that a minimalistic (and growing) deliverable will be consistently present throughout the project timeline.

6.1 Vision Pipeline

The vision pipeline described above was partially functional before the concept of gameplay came up. The original use of this process was the segmentation of real-world objects. A number of previously gathered datasets were collected to test the pipeline. Among these datsets were those gathered by Dr. Alexander Stoytchev for his dissertation. the mentioned dataset consists of a robotic arm with a large colored marker at each joint. This dataset maintained most of the assumptions necessary for the vision pipeline to work. That is, most of the movement was two-dimensional and orthogonal to the field of vision and the relevant element had sufficient texture to be picked up by OpenCV feature tracker (as is often the case for real-world objects). Segmenting this image based on components as described



(a) pong (b) SDLball (c) Arkanoid_SDL

Figure 10: The three iterations of games as they were perceived throughout the process. The pong image (left) was the ideal image for the image as was scene in the mind of the developers as the beginning of the process. SDLball (middle) was an open source game that was stripped down and piped frame by frme into the vision processing. The actual Breakout game (right) was made from scratch and named Arkanoid SDL. It incuded a white background and larger, roughly textured objects.

here proved quite effective. All the individual joints of the robotic arm were being segmented with reasonably minimal variability.

6.2 Gameplay with Shared Memory

When the first game concept was introduced, effort was put forth to replace the stored dataset with one that is gathered live from the game. Ideally, this could be done without introducing the complexity associated with video noise (reflections, etc). A system was, therefore, set up that piped game visual data directly into the grouping algorithm via shared memory and networking. With this system in place, a preliminary game-playing system could be implemented.

6.3 Joystick Controls

The Joystick controls were not significantly altered from the first time the Joystick manipulation algorithm was developed. The one notable change involved temporarily increasing torque limits on the arm joints for pre-exploration then decreasing them back once the pre-exploration step was completed.

6.4 Game Iterations

The introduction of real-world vision issues required the games and to change between iterations. The most notable such changes were The increase in size of the relevant components and the change of the background color to white. The increase of size was necessary as a significant amount of vision real estate was now taken up by elements outside the game. The background change was necessary to reduce the effect of reflections on the tracking algorithm.

7 Experimental Results

7.1 Vision Process

As mentioned before, its important for the vision model to maintain a consistent analogue of the individual game elements in order for the self-other to perform with any significant fidelity. The most telling metric for the performance of the visual model is the length of the lifetime of each of the components. When a component loses too many point features, it goes out of scope and is impossible to recover with the current system. A dataset was gathered over 5406 Frames with the final iteration

of the Arkanoid game. The arkanoid game has two distinct moving elements (the paddle and the ball) and should, therefore, ideally have two components moving around the screen. The final component count was 253 which sounds bad at first but is not really telling of the system performance.

If the component lifetimes were distributed with a symmetric distribution, then components would be expected to last about 20 frames each. With a frame rate of ten to fifteen frames per second, that comes to a pretty miserable performance. A passing glance at this distibution reveals that it's heavily skewed.

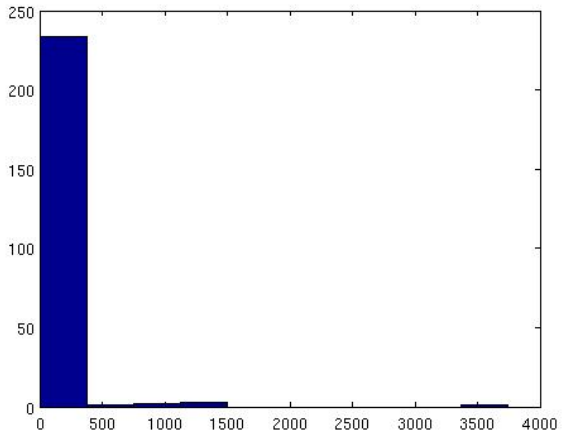


Figure 11: The distribution of component lifetimes for a dataset collected on the Arkanoid_SDL game through the robot's eyes over 5406 frames

Clearly, this ditribution is very heavily skewed. This is caused by the vast number of components that appear for one or a handful of frames then disappear. This is essentially noise and can be somewhat easily filtered from the rest of the system. Dismissing the noise yields a perhaps more intelectually ineteresting result.

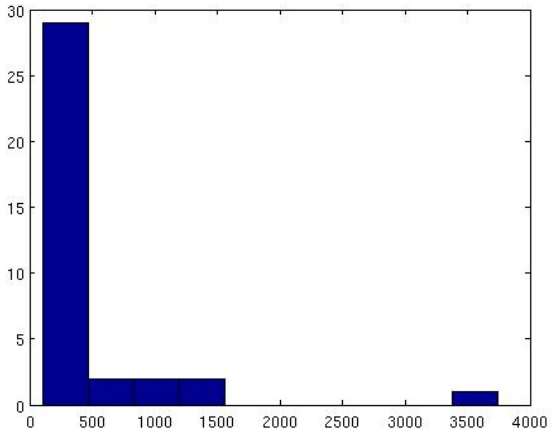
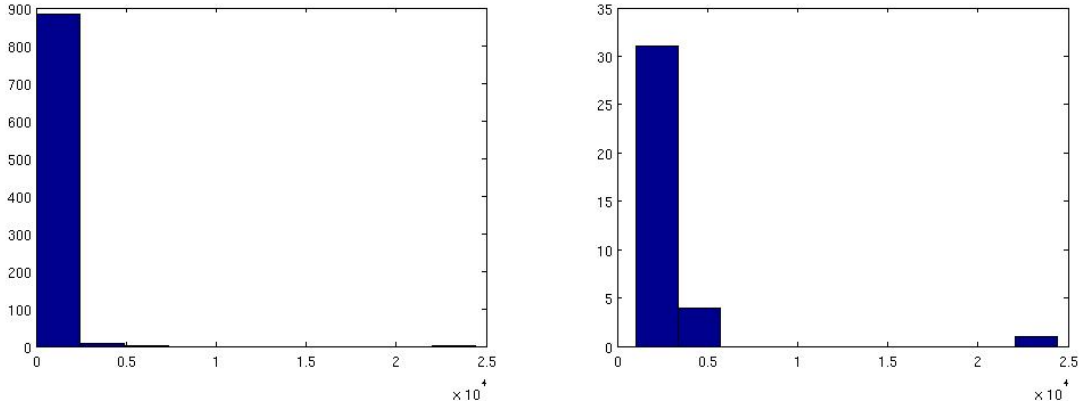


Figure 12: The distribution of component liftimes along 5406 frames with any component that lasted longer than 100 frames (8 seconds)

These numbers are still thoroughly skewed for the worse but there are a few components that lasted for a significant portion of the sample time (a handful of minutes). A minute is a very significant time and would provide enough contingency information for even a very slow self-detection algorithm.

As mentioned before, this result was based on data gathered using the webcam and the final iteration of the game. It was seen as necessary to explore a cause of these mediocre results. This type of collection was then performed on a dataset with more ideal visual qualities. there result is given here.



(a) Complete Results

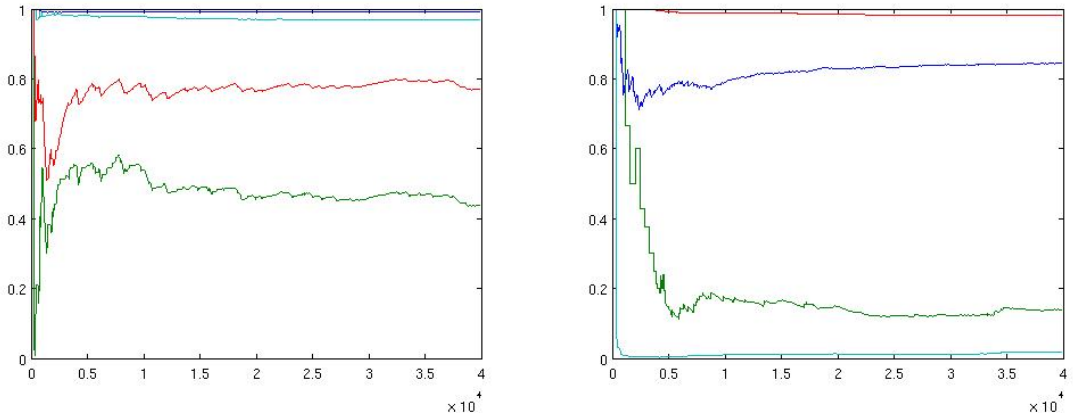
(b) Results Filtered for Noise

Figure 13: Distributions of component lifetimes for images piped into the game from shared memory

This data was gathered at an earlier iteration of the system in which the game images were piped into the system directly via shared memory. During the implementation of this project, the robotics platform got notably more fragile. This earlier version, therefore ran much longer and this data was gathered over 38766 frames. As before, this created a huge amount of single-frame components that cloud up the data. The zoomed in data shows a much better result, however. 36 components lasted over 1000 frames (well over a minute) and one lasted 24423 frames which is the majority of the system runtime and far beyond the minimal point at which significant data can be gathered about the component.

7.2 Self-detection

The the performance of the self-detection algorithm has two valuable parameters. The accuracy of the self-detection algorithm is a trivial metric as literally all of the components that survived for a nontrivial number of frames converged on the correct self/other classification. The other important metric is the convegence time. As stated before, the component identities are somewhat fragile making it necessary for their identity to converge as quickly as possible. Results are displayed here.



(a) Self Component (b) Other Component

Figure 14: The results for Self-Other separation. These graphs demostrated that the true reliable metrics are sufficiecy metric (blue) and the inverse necessity metric (teal). Since the ball in this example was almost moving, the inverse necessity metric was essentially 0 and the forward necessity metric was essentially 1.

All components that had a significant lifetime displayed convergent behavior very similar to that shown here. A notable result is the rate at which these components converge on their respective identities. Within approximetly 50 frames (about 4 seconds) both components were locked in correctly as "self" or "other".

7.3 Gameplay

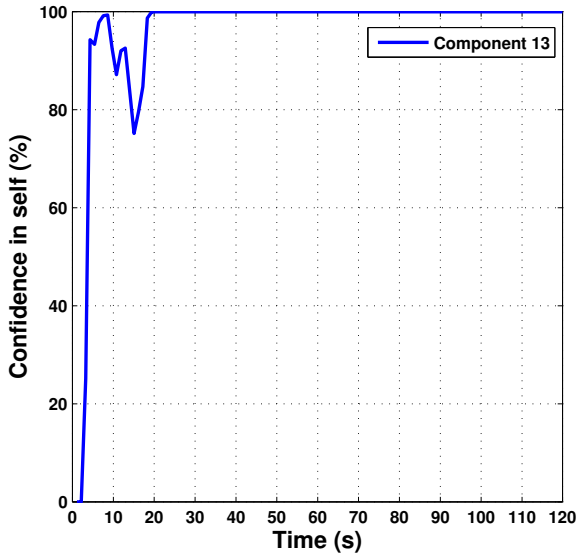
The physical limitations of the system and the performance-intensive nature of these games made this system largely inadequate to play these games. After signifiacnt babbling, the system was sufficient to deflect the ball with approximetly 60% accuracy. As far as the games go, this isn't enough to keep the ball afloat more than half a dozen times making this system ultimately uncompettive.

8 Summary and Future Work

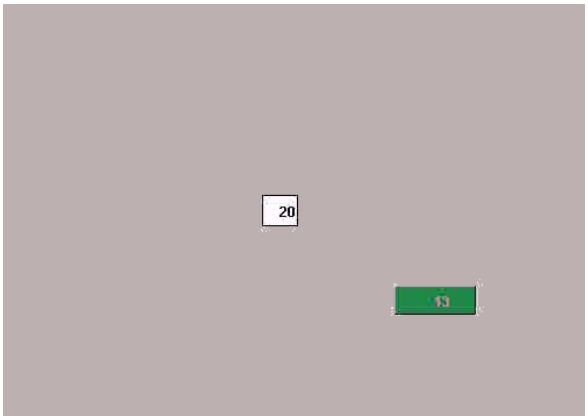
As was mentioned previously in various part of this work, the modular design of the project allows for individual components to be extracted, optimized, and placed back into the system without necessarily effecting the other systems. Some of these potential changes were in a state in which the technology has been too far from implemented to operate in the real-time system described here. Other changes were more or less implemented but were either scoped out of this work or followed a mindset that was simply too different than one undertaken here.

8.1 Entropy-driven self-detection

A project being peformed in parallel to this one performed some tests highly correlated to the work here. Among the data processed was that produced by the visual model described here. The components, time stamps, and motor commands were used to test self-other separation for a short time period using this entropy-driven model. The results were, overall, promising. At the time of publication, this self-detection process was implemented in a way that makes it viable to adapt for this process.



(a) Entropy-Driven Self-detection Results



(b) Entropy-Driven Self-detection Visualization

Figure 15: An alternate approach for self-detection was introduced late into the creation of this process. This Entropy-Driven model showed promising results but was too slow in both runtime and convergence time.

The rate of convergence was, however, seen as too slow in order effectively play these games given the somewhat incosistent results of the vision process.

8.2 Improved Feature Detection and Optical Flow

Alternate forms of feature detection and Optical flow functionality were considered extensively. Most of the alternative feature detectors were quickly found ineffective in producing corner features that were easily trackable from frame to frame. Features found using other detectors were also added to those found from *cvGoodFeaturesToTrack* but this addition didn't add any significant stability and simply created extra noise in the vision model. An implementation of optical flow from Brown University was also tested with images gathered during this process. The optical flow actually produced very promising results but the algorithm itself was found to be far too slow (by a factor of about 1000) to effectively work in a real-time system.

References

[1] V.Sukhoy, J. Sinapov, L. Wu, and A.Stoytchev, "Learning to Press Doorbell Buttons," in Proc. of ICDL, 2010, pp.132-139

[2] E. Klingbeil, B. Carpenter, O.Russakovsky, and A. Ng, "Autonomous Operation of Novel Elevators for Robot Navigation," in Proc of ICRA, 2010, pp. 571-758

[3] M. Williamson, "Rhythmic Robot Controls Using Oscillators," IROS 1998

[4] A. Saxena, J.Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," The International Journal of Robotics Research, vol. 27, no. 2, pp. 157-173, 2008

[5] M. Hillman, K. Hagan, S. Hagan, J. Jepson, and R. Orpwood, "A Wheelchair Mounted Assistive Robot," in Proc. of ICORR 1999, pp.86-91, 1999

[6] S.E. Salcudean, G. Bell, S. Bachmann, W.H. Zhu, P. Abolmaesumi and P.D. Lawrence, "Robot-Assisted Diagnostic Ultrasound - Design and Feasibility Experiments", MICCAI'99, Second Intl. Conf., pp.1063 - 1071, 1999

[7] G.H. Ballantyne, "Robotic Surgery, Telerobotic Surgery, Telepresence, and telementoring: Review of Early Clinical Results," Surg Endosc 10:1389, 2002

[8] J. McCarthy and P.J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," Machines Intelligence 4, 1969

[9] M. Newborn, "Kasparov vs. Deep Blue: Computer Chess Comes of Age," 1997

[10] G. Tesauro, "Programming Backgrammon using Self-teaching Neural Nets," Artificial intelligence, 134 pp. 181-199, 2002

[11] D. Billings, A. Davidson, J. Schaeffer, and D Szafron, "The challenge of poker," Artificial Intelligence 134 pp.201-240,2002

[12] M. Genesereth and N. Love, "General gameplay: Overview of the AAAI competition," AI Magazine pp.26

[13] H. Huang and C. Liang, "Strategy-based Decision Making of a Soccer Robot System Using a Real-time Self-organizing Fuzzy Decision Tree, Fuzzy Sets and Systems," 127 pp.49-64, 2002

[14] J. Kim, H. Shim, H. Kim, M.Jung, I. Choi, and J.Kim, "A cooperative Multi-agent system and its real time application to robot soccer," in Proc. IEEE Intl. Conf. Robot Automat., April 1997, pp.638-643

[15] D. Povenielli and J. Cant, "Arboreal Clambering and the Evolution of Self-conception," The Quarterly Review of Biology, 1995

[16] J. Watson, "Detection of Self: the Perfect Algorithm," Self-awareness in animals and Humans: Developmental Perspectives pp.131-148, 1994

[17] P. Michel, K. Gold, and B. Scasselati, "Motion-Based Robotic Self-Recognition," proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, 2004

[18] A. Stoytchev, "Self-detection in Robots: a Method Based on Detecting Temporal Contingencies," November, 2010

[19] R. Saegusa, G. Metta, S.Sandini, and S. Sakka "Active Motor Babbling for Sensorimotor Learning," Robotics and Biomimetics, pp. 794-799, 2008