# Using Linear Splines to Continuously Develop Understanding of Affordances

Erik Lagerstedt
drattans@gmail.com

*Understanding affordance is a bottom-up process that lets simple visual features be used as clues on how to use an object, so that reasonable interactions with objects can be achieved, without any semantic knowledge. Linear splines are proposed as a way for robots to fast and continuously learn how their actions affect their surroundings. The general conclusions that they can draw from only little experience, using this method, are seen as an example of understanding affordances, and the specific case investigated by the robot created as a part of this thesis, is how pushing actions affect small blocks. This information will be used by the robot to perform goal directed tasks, namely to push the block to specific predetermined positions. A way to make generalised predictions, so that actions on new shapes can be performed in a reasonable way without any previous interaction, is proposed. It becomes obvious that good inner representations of actions and reactions are necessary, as well as some management of learned relations.*

## 1 Introduction

In humans, visual input is collected by the eyes and then mainly sent to the occipital lobe in the back of the head. From there it is primarily passed through two routes. The ventral route, leading into the temporal lobe, is commonly called the *what stream* and it is associated with semantic understanding of objects in view. The other (dorsal) route leads into the parietal lobe, and it is commonly referred to as the *where stream*, since it handles the location of the objects in view (Ungerleider & Mishkin, 1982). Goodale and Milner (1992) proposed that the dorsal stream uses information which is separate from semantics, but still helpful when an action is attempted to be performed on an object. This definition is wider than just where an object is, and according to them, the dorsal route may also handle simple features such as shape, size and colour, which can be used as clues to what is possible to do with the object. This route is thus sometimes referred to as the *how stream*.

When performing an action on an object, the importance of the object's location is trivial. The importance of other features related to the how stream, might not be as obvious, however, these features, which are simple and general (e.g. concave or fist sized), are used to determine something commonly referred to as "the affordance of the object." Separation of affordance from semantics have been used by Hodges, Bozeat, Lambon Ralph, Patterson, and Spatt (2000) to explain how people suffering from semantic dementia are able to use tools, even though they cannot explain what the objects are. Lauro-Grotto, Piccini, and Shallice (1997), also examining people suffering from semantic dementia, report that the patients can perform relevant behaviours when presented to visual (but not verbal) input of objects. This is another strong argument for affordance being separate from semantics.

Affordance is a concept that was coined by Gibson (1977), and it is inspired by the Gestalt psychological idea, that objects are perceived as their properties, more particularly how objects can be used. This is a bottom-up process which, for example let you see an object as gripable, even before you realise that it is a door handle. The initial definition was that "the affordance of anything is a specific combination of properties of its substance and its surfaces taken with reference to an animal" (Gibson, 1977, p. 67). The last part emphasises that affordances of an object can vary depending on which agent is perceiving it, however, this does not mean that affordances are subjective. In this sense it is similar to the umwelt-concept, which discusses the issue that the properties in the world which are relevant depends on things such as the physiology and size of the agent (Uexküll, 1957). This general description of affordance might seem clear, however, the strict definition of affordance has been changed many times, and it is debated to this day. Gibson is said to never have been satisfied with any definition, and he liked that it developed continuously (Şahin, Çakmak, Doğar, Uğur, & Üçoluk, 2007).

The concept has been studied from the perspective of many fields. It is central in ecological psychology and support of it has been found in neuroscience. Most research is unfortunately only dealing with the perception aspect of affordance. Other interesting areas, such as uses in learning or high level reasoning, are still unexplored to a large extent. Exceptions to this are for example Fitzpatrick, Metta, Natale, Rao, and Sandini (2003) and Stoytchev (2005), who train robots to map actions to consequences, or MacDorman (2000) who train robots to determine affordances of its body and surrounding, and then uses it as a tool for navigation. Çakmak, Doğar, Uğur, and Şahin (2007) use affordances of the agent-environment system, mentioned below, and formalise it to be (*effect*, (*entity*, *behaviour*)), with which they mean that a certain effect is achieved after a specific behaviour is applied on an entity. With this formalism it is possible for the robot to create lists of relations between the three, as it explores the world. Their robot manages eventually to develop goal-directed planning and navigate in an environment with obstacles, without using simulation, and with minimal use of visual input. This was later developed, so that multi-step planning was possible, even though simple internal simulations were used in this case (Uğur, Şahin, & Öztop, 2009).

When the concept of affordance is attempted to be formalised, one of three perspectives are used: the agent's, the environment's or the observer's. As an example, consider a scenario with a human watching a dog playing with a ball. The dog is in this case the agent, and affordance is always in relation to the behaviour of the agent. The agent may push the ball, which means that the the

ball is pushable. The ball represents the environment in this example, and affordances from this perspective are how it can be manipulated by its surroundings. Properties such as be pushed by the agent, and not seen by the observer are possible affordances. The human is, as a third part, the observer of the dog-ball system, which, as an example of affordance, can be said to have the ability to push (Şahin et al., 2007). All three perspectives are commonly used, however, it is not always clear which one in the different cases. The resulting confusion is one reasons why the debate on the definition of affordance is still ongoing.

Action is an important part of cognition. It serves not only as one of few output channels, but it also offers ways to manipulate the world, so that cognitive processes are facilitated. The act of manipulating the world in this fashion has been proposed not only to serve as preparation, but also being a kind of cognition in itself (e.g., Kirsh and Maglio, 1994). To what extent semantic knowledge of an object is needed, for the object to be used, is debated. A simple model would suggest that semantic understanding includes information on how an object is used, which would make semantics crucial to use the object. However, as mentioned above, there are empirical evidence against this simple model (Hodges et al., 2000; Lauro-Grotto et al., 1997), which indicates that semantic understanding is not necessary when detecting affordances and using an object. If only semantic understanding is available (that is, if parts crucial to the how stream have been damaged), Rossetti et al. (2005) found that slow, off-line strategies are used when performing actions as responses to visual input.

In this thesis, a block pushing robot arm (henceforth commonly referred to as "Radagast" (Robot Arm Developing Appreciation for General And Specific Traits)) will be used to explore how affordance can be approached by machines. The knowledge of the affordance of objects will be used to perform goal directed actions, without the robot having any semantic understanding. The result might be generalisable to other agents, however, whether that is true or not will not be pursued in this project. The robot will, more specifically, learn to push blocks to predetermined goal positions, and it has to find the relation between a push action and the movement of the block, by trial-and-error. The problem has been simplified to two dimensions by using the angle from where to push as input, and the angle to where it moved as output. A push will always be towards the centre of the block, and it will always be intended to make the object move towards the goal position. The objects are assumed to be of constant apparent size, and constant distance from the camera with which the robot sees. This is not entirely true, and deviation from these assumptions will result in noise.

When the robot manages to handle one object, other shapes are intended to be introduced to establish a new problem for the robot. A technique of pushing that was previously confirmed to work flawlessly, is now useless. The flawlessness and uselessness are strong words in the context, and what it means is that the noise in the knowledge should be smaller than the effects of the new shapes. The shape of the block then has to be taken into consideration when mapping a push and the reaction by the pushed block. A set of strategies can be created, which the robot can choose from, using an objects shape as a cue.

When the second shape is introduced, the knowledge of the first shape can initially be copied, and then modified over time. Each new shape could then start with the knowledge from the latest shape, however, that would not make much sense. A better idea would instead be to create an incomplete and abstract prototypical shape. It can be introduced as a copy of the first shape when the second shape is introduced, and then be modified so that it holds information that works for most shapes. To use this as a base when a new shape is introduced would make sense since it probably will contain information on general behaviour, and only the shape specific traits must be learned. This could imaginably happen in several levels as well, so that new prototypes can be introduced, but have a more general prototype to be based on. This would open up for the possibility to use some abstract shape space to predict techniques for new shapes. If circular and edgy are the two known mid-level-prototypes, then a triangle shaped block might copy the edgy prototype, and a square shaped block might try to find some way to be based on some combination the two prototypes. This feature would require high fidelity and large differences between the action-reaction-functions of the different shapes.

The objects in this thesis can, from Radagast's point of view, have different kinds of pushability as affordance. These pushabilities will be represented as functions, relating block movement to push actions. To approximate these functions, linear splines will be used, that is, between each adjacent pair of meassured data points a linear function will be fitted.

# 2 Materials and methods

*The machine*

The robot (see figure 1) consists of four dynamixel AX-12 servos, creating an arm with four degrees of freedom (DOF). Since the arm always will keep its fingertip somewhere in a plane (parallel to the surface on which it stands), the software only uses the two relevant DOF, which in the end is converted to angles of the servos, using inverse kinematics. The robot can reach places between 9 cm and 24 cm from the base of arm.

The input to the robot is the visual (RGB, later transformed to rgI) input from a Microsoft Kinect-device, suspended 50 cm above the surface. This will give an overhead-perspective of the experiment for the arm to work with. Near the edges of the image, the perspective is slightly from the side, which is a source of error. Functions to compensate for this could be introduced, however, the caused errors were not deemed severe enough for that to be prioritised. The robot will also use proprioceptic information, in form of feedback from the servos.

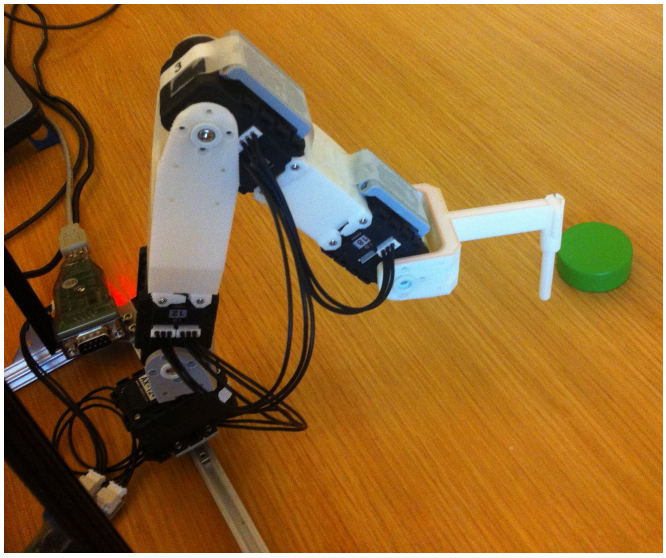

**Figure 1:** A photo of Radagast, the robot used in this project. The green object near the finger of Radagast is one of the pushable blocks.

The pushable objects are thin plastic cylinders (diameter 4.2 cm, thickness 1.5 cm) laying flat on the surface, and one of them can be seen in figure 1. Square and triangular shaped blocks of similar sizes are also introduced. Their colours are distinctly different from the surrounding, to facilitate the tracking of them.

*The system*

In Ikaros[1], which is used in this project as the framework, code is implemented as modules using the programming language C++, and the modules are then connected using xml files with a description of the system. Several features, such as an interface for the servos, and colour discrimination algorithms, are already implemented as modules. To learn the relationship between arm movement and block movement, a new module, implementing a linear spline function, was created. Other functions, such as a shape detection module, were also implemented as the need for them arose, and they are listed and discussed below. Figure 3 sketches which modules are used, and how they are connected. The information flow starts with the input image from the Kinect device, and ends with arm movements via the Dynamixel servos. The system has, in figure 3, been split into four sections to facilitate a discussion.

The idea is to let the robot predict what action will get the pushable block closer to the goal position. If the predictions are too wrong, it might indicate that the object has a different shape than anticipated, which could encourage Radagast to assume that some specific rules might be relevant for this object. This latter feature has not been implemented, since there are other sources of error that are much more dominant than the error that could be used as a cue. This will be discussed more in section 3 and 4.

**Sensor** The first section contains only one module, which is the Kinect module. It works as an interface between the Kinect-device and the system, and the only purpose of this module in Radagast, is to take the video image from the Kinect, and forwards it as an image of 640x480 RGB encoded pixels, into the system. This module is part of Ikaros.

**Image preparation** All the modules in this section are already part of Ikaros. The first module (ColorTransform) takes the input image and transforms the colours from RGB to rgI. This is to make the image less sensitive to changes in light conditions. ColorClassifier defines a colour subspace and finds every pixel with a colour in that region. The output is an image of the same size as the input image, however every pixel now has the value one or zero, indicating whether or not the the pixel had an appropriate colour. The final module in this section (SpatialClustering) takes the image from ColorClassifier and recognises it as a monochrome image. In this image, clusters of white will be identified, and their locations will be sent as the output. This method of finding coloured object is more thoroughly described by Balkenius and Johansson (2007).

**Core** CropMod will take the monochrome image from ColorClassifier and the cluster locations from SpatialClustering, and create an image of 100x100 pixels. This image will be a copy of the part of the input image which is centred around the cluster centre.

The first thing that happens in ShapeMod is that the input image from CropMod will be run through a dilation algorithm, to remove empty pixels in the white cluster, which sometimes occur due to noise. It is not necessary to removed them, but doing so increases the quality of the image slightly. If no shapes are known to the robot (which will be the case in the beginning), or if some input indicates that more shapes needs to be known, this module will use the current input image as a template. Learning a shape means, in this module, that the image is rotated, so that information on how the shape looks from different angles is learned (see figure 2). The original image, along with the rotated versions of it, are stored in a list.

If there already are shapes in the list, this module will compare the input image to all the stored images, to find which shape and orientation fits the best. It will also try several locations of the template, to see if the centre of the cluster in the input image really coincide with the centre of the object. The definition for the centre of the object from the SpatialClustering module is the average position of the pixels in the cluster. This is a good definition if the entire object is seen, however, if part of the object is obscured, an offset will appear. This module will compare slightly shifted images to the input, to find a better fit. Both a greedy search algorithm and a simulated anneal-

**Figure 2:** An example of an image of a shape being rotated.

[1] http://www.ikaros-project.org/

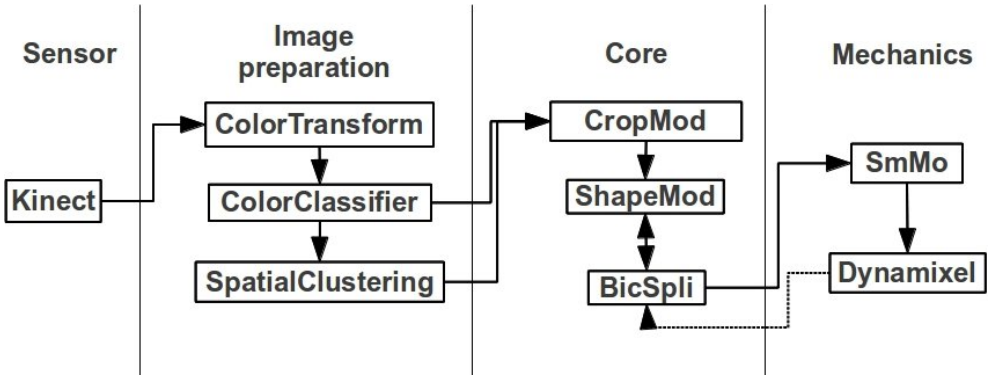


**Figure 3:** A sketch of the structure of the system. The flow of information starts with the Kinect module and ends with the Dynamixel module. The system has been divided into four different sections. The sensor section contains the Kinect module, which provide images from the outside world. The image preparation section finds the interesting features in the image and passes it on to the core, which performs most of the work in the system. Ultimately the core will produce coordinates of the desired position of the finger, and the mechanics section will use inverse kinematics to find the appropriate angles for the servos, which will finally move. The dashed line in the figure represents feedback.

ing algorithm were tried when searching for the best position. A few different sets of parameters were tried, however, these algorithms were not successful enough with any of them. Using a brute force approach, better approximations for the object locations are usually found. The algorithm is relatively slow, however, since the shift is very rarely more than ten pixels in any of the four directions, that can be used to limit the search space to something manageable. The index of the shape, the direction of the shape and the location of the object are sent as output.

The purpose of the BicSpli module is to decide how the arm should push the block. The first thing that happen is that the direction to the goal position from the block is found. This will be the direction in which the block will be intended to move. After it is found, one of four modes are used, the four modes being preparation, evaluation, transportation and execution.

1. In preparation mode an angle from which the robot should push the block, is determined. The robot will, if possible, use past experience to decide this, and will otherwise choose an angle randomly.

2. In evaluation mode it will compare the results of a recent pushing action with the predictions that was made for the same pushing action. It will then update its knowledge in an appropriate way. This will be discussed more extensively below.

3. In transportation mode, the location of the start of the push is calculated and sent to the next module. The push angle produced in preparation mode is relative to the object, but the modules later in the chain uses coordinates relative to the base of the robot. Converting the coordinates from the first to the latter, is an important part of this mode. While the finger moves to the instructed position, it should not touch any object, since that would change premise. To avoid that, the finger will be slightly elevated, while in this mode, so that it will pass over any potential blocks in the way.

4. In execution mode instructions on where the push will end is sent to the next module. Entering execution mode, the finger will be lowered from its levitated state. Other than that, this mode works much like transportation mode.

The modes are commonly cycled like this: the start of the push is decided in preparation mode, and transportation mode is engaged. It will stay in this mode until the proprioceptic feedback indicates that the arm is in the right position. The finger will be lowered as it enter execution mode, in which it will stay until the push action is complete. The robot will then enter evaluation mode to evaluate the effect of the push.

The knowledge that is acquired is stored in lists. The information that is stored is from what angle $y$ the block was pushed, and the corresponding result $x$. The result is seen as a function of the angle from where the pushing started, $y = f(x)$. Between all the perceived cases, linear functions are approximated:

$$f(x) = \begin{cases} m_1 x + k_1 & \text{if } x_n < x < x_1 \\ m_2 x + k_2 & \text{if } x_1 < x < x_2 \\ \vdots & \vdots \\ m_n x + k_n & \text{if } x_{n-1} < x < x_n \end{cases} \quad (1)$$

where $(x_1, \ldots, x_n)$ are the different experienced results of the tried push angles $(y_1, \ldots, y_n)$. The constants $(m_1, \ldots, m_n, k_1, \ldots, k_n)$ are calculated using the push angles and their results. Constructing a function from data points like this is called using linear splines. The first condition in equation (1) will not only cover angles smaller than the smallest tried angle, but also angles larger than the largest tried angle. The notation might be a bit confusing, however, if the cyclic behaviour of the angles are considered it should make more sense. Using these approximated functions, it is possible for the robot to make qualified guesses about the reactions to untried push actions. When Radagast decides how to push, the desired result is inserted into $f(x)$, and the value of the function in that point will be used as push angle. Figure 4 is an example of how this function can look.

Apart from saving actions and reactions, information on whether or not the robot trusts the different parts of the data is saved. If a push action is performed, with a result too different from what is predicted by the fitted function, that segment of the function is declared unreliable, and is no longer used when searching for new start points for future push actions. A randomly chosen point is used, rather than using unreliable predictions. Whenever

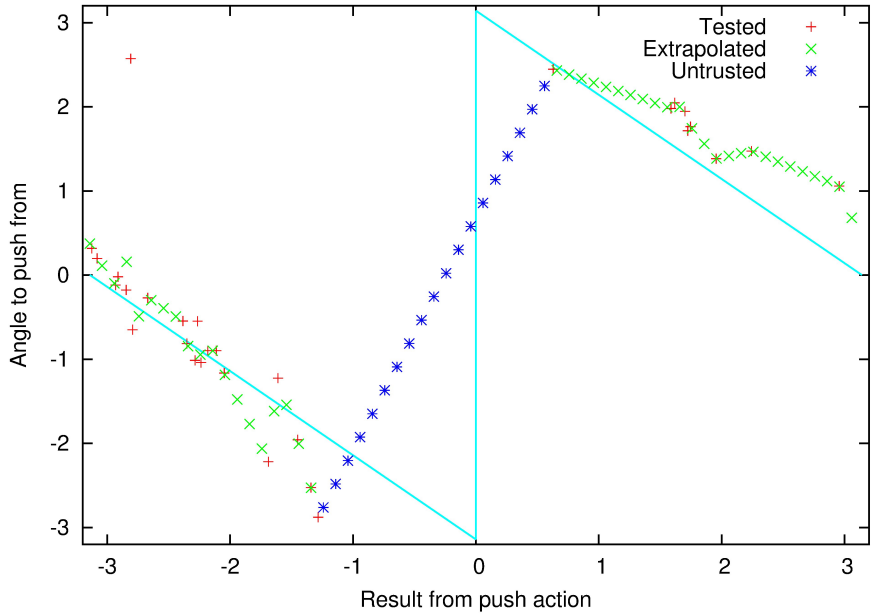


**Figure 4:** An example of how the approximated function can look. The red crosses are experienced instances, and the other crosses are probings of the fitted function. The blue crosses are in regions that are deemed untrustworthy. The cyan lines are the correct function. These guidelines have been added after the robot is done, so it will not be able to use them.

a new point is found in an unreliable interval, the function will be updated and the region will be trusted again. If it is a wide region, random angles will be needed to be used often, however, it will also be very likely that one of the randomly chosen points will lie within the region, so it will not take long for this region to be updated. Narrow regions will more rarely be updated, since it is less probable to choose an angle within it. On the other hand will a narrow region probably not be needed so often. Even if the narrow region turns out to be important, it should be easy for the robot to push the block away from it, using randomly chosen points. The ability to not trust parts of the data can sometimes be very important to avoid getting trapped in a nonsensical series of actions.

The robot is learning continuously when it is running. It is not, however, remembering every action-reaction-pair that it has ever encountered. Unlike what is common in the field, the robot not only learns when it is successful. The robot saves the information if the new data point will be in a region that is considered unreliable, if the reaction diverges too much from prediction, or if it falls within a large region without any data points. The last reason is primarily to reduce the importance of outliers. If the correct function is found quick and with few examples, the robot will otherwise only learn noisy points, which will have a large effect on the function, since their relative frequency compared to low-noise points, will be high. Another problem with large regions is that if some part of it is unreliable, the entire region will be useless.

If something indicates that a new shape is needed to be learned, this module will send a signal to ShapeMod, and then create a set of lists for the new shape. When the second shape is created, the information from the first shape is copied, and not only used as a start point for the second shape, but also saved as a prototypical shape. Any shape created after this will start with the information of the prototypical shape, so that general pushing behaviours are not needed to be learned repeatedly. Whenever an action-reaction is experienced, that seems to fit the expectations of most known shapes, the prototype will be updated. One catch with this is that the rule for when a new shape is needed, is not yet implemented. The learned data is at the moment considered too noisy to do that in a good way, and this will be discussed more in section 4.

**Mechanics** The desired position of the fingertip will be accepted as input for the SmMo module. By using inverse kinematics, the angles for the servos is computed and sent as output. The path is divided into small enough parts for the transportation to be fairly smooth. This is not only an aesthetic detail, since too rough movements might damage the robot or its surroundings. The module will discriminate between execution mode and transportation mode, and the difference is that the fingertip will be slightly elevated in the transportation mode, to avoid the block being pushed by accident. The path will always be the shortest possible, which makes it possible for the arm to get stuck. This can for example be if the shortest path is through a region in which the robot cannot reach, even if there is a simple ways around it.

The last module, Dynamixel, will handle all communication with the servos. It will take angles for the servos as input, and return information on their actual position. The returned values are used by BicSpli as proprioceptic information. This module is part of Ikaros.

# 3 Experiment and results

*The experiment*

To find out how well Radagast performed, the time it took for it to get a block from a location to the goal position was measured. The unit used for measuring the time was number of ticks, which is the number of time each

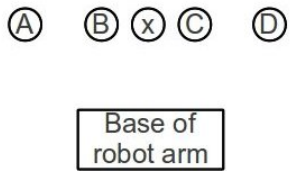


**Figure 5:** A sketch of the setup of the experiment.

module have been called. Before restarting, the block was placed in a new position, to see how well the acquired information could be used. The start positions were always the same, however, due to the random angles that sometimes were used by the robot, different finishing times are expected in the different trials. In all the trials in this experiment, the circular shaped block was used.

Figure 5 sketches the setup. The first start position (B) was 3 cm to the left of the goal position (X), and when the robot had finished, the block was moved to the second start position (D), which was 12 cm to the right of the goal position. Ten trials were conducted like this, and then another set of ten trials were performed, with the difference that it was mirrored. In these trials the first start position (C) was 3 cm to the right of the goal position, and the second start position (A) was 12 cm to the left of the goal position. If the task took more than 2000 ticks to solve, that part of the trail was cancelled.

*The results*

The time it took to perform the different tasks are reported in table 1. If the robot did not succeed within 2000 tick, it was timed out, which explains why some times are $>$ 2000 rather than a specific number. The small number of trials makes it very hard to find any statistically significant results, and regardless of the results of the statistical tests, the power of them would be very low. The only statistical test performed was a $\chi^2$-test to see if the difference in number of time-outs in case D, compared to case A, were significant. The reason to test this was because it seems to be the clearest result from the experiment, and it should thus have the best prospects for a significant p-value. However, when compensating for the small data set, using Yates' continuity correction, it turned out that not even this result was significant (p=0.072).

**Table 1:** The numbers in this table is the time it took (measured in ticks) for Radagast to successfully push the block from the different start positions (A, B, C or D) to the goal position (X). Column B and C represents the start positions close to the goal X, and D and A the start positions far from the goal X.

| B | D | C | A |
|---|---|---|---|
| 207 | 1606 | 42 | 945 |
| 158 | >2000 | 392 | 737 |
| 53 | >2000 | 213 | 327 |
| 253 | >2000 | 367 | 235 |
| 122 | 50 | 112 | 433 |
| 250 | >2000 | >2000 | >2000 |
| >2000 | >2000 | >2000 | >2000 |
| 116 | >2000 | 217 | 278 |
| 101 | 1168 | 1053 | 1402 |
| 197 | >2000 | 1494 | 661 |

By inspecting the numbers in table 1, it is possible to see a problematic trend. The issue is the apparent bias in the push direction. The median times when pushing left is 380 ticks in the short case (when pushing from C), and more than 2000 ticks in the long case (when pushing from D). This is more than the corresponding 178 ticks and 699 ticks respectively when pushing right (from B and A respectively). The success rates are also lower when pushing left compared to when pushing right (0.8 compared to 0.9 in the short push case, and 0.3 compared to 0.8 in the long push case). The reason for this bias is the discontinuity in the function relating block movements to push actions. This discontinuity is, in turn due to an unsuitable choice of representation of directions, and it will be discussed in section 4.

Because of this bias, which is not due to a physical restrain, but rather a weakness in the systems representation of the world, there are no reasons to continue with further experiments. This issue needs to be sorted out before any interesting phenomena can be studied. This is reason for the experiment being so small, and for only using one kind of the block shape in it.

# 4 Discussion

Radagast has a hard coded low-level feature detection ability, which without any semantic knowledge, finds pushable object. This only works if the pushable objects have a distinct enough colour compared to the surroundings. By learning the relation between push actions and movement of blocks, it is possible for the robot to move blocks to goal positions. Throughout this project, the goal positions have been constants, however, they are not defined in any of the systems modules, but rather feed as input to the interested modules. If other modules, of higher cognitive level, can be constructed to perform planning and reasoning to produce new goal position, they can easily be used by this system. This system only tries to push objects towards the goal position, but other paths can be achieved if necessary by constructing subgoals along that path. In this way, this system can be used as a low level function, which uses simple features in visual information to solve simple object manipulation problems which can be used by higher level systems. As the robot gains experience by interacting with its surroundings, a sense of what behaviours are appropriate to use on different objects should appear. This is what affordance does for many animals, but a few problems lingers in this system, preventing it from clearly showing understanding of affordance. This discussion will primarily revolve around that subject.

For simplicity, the robot always tries to push towards the centre of the block, so the only input variable is an angle describing from what direction the push is coming, and the value of this angle can thus vary from zero to two pi. Since $0 = 2\pi$, the function should continue from zero if it exceeds two pi. This has been taken into account when the function that relates block movements to push actions have been defined, by letting the linear function before the first point use the last point when being fitted,

and vice versa (see equation 1). The other angle in the function is a very simple way of representing the resulting movement of a block after a push action. It became apparent in the experiment, that there is a discontinuity in the function, since $0 = 2\pi$, and that this is a problem. The discontinuity can be seen in figure 4 as the vertical cyan line at $x = 0$. In the same figure, the only untrusted region (sketched with blue stars) is around this discontinuity, which is a common phenomenon. It is hard to learn how this region behaves, and even if good results are achieved, even the slightest amount of noise around this point is often enough to destroy it.

One way to solve this problem is by changing the representation to sine and cosine of the angle, which would remove the discontinuity. That would, however, enlarge the problem space from two to at least three dimensions. That is not preferable since problems with higher dimensions will take much more time to handle, and more examples would be needed before conclusions can be drawn by the robot. Linear splines will also be inappropriate for problems of higher dimension. In three dimensions, bilinear splines could be used, however, when the number of dimensions increase, it might be better to search for ways to solve this problem that are more adapted and specialised for higher dimensions. Considering the result of the experiment, there are little choice regarding giving up the angle versus angle representation.

The second largest problem is more of a mechanical issue. When Radagast is experimenting, it tends to push the blocks to places where it cannot reach them. When this has happened, the block has simply been picked up, by a supervising human, and placed back in the region that is reachable to the robot. There is little to do to prevent the robot from sometimes pushing the objects out of reach, unless boundaries are introduced as a new concept to be learned. Boundaries are interesting, but probably deserves a project of its own, so it might not be preferable to attempt before better results are achieved with (seemingly) unbounded pushing. Letting a human return the blocks might, however, introduce bias, or other unwanted information in the learned data. A related issue is that Radagast never considers the possibility that someone else might move the object. If this happens while the robot is in the wrong mode, it will learn very strange behaviours. A necessary feature, that needs to be added, is thus the ability to prune the remembered data. Incorrect data points are bound to accumulate in the memory, especially since it is primarily unexpected events that are remembered. Even if there are ways to improve the modules, to reduce the number of strange events, there are always undesirable things that will happen, and noise is a natural thing to expect. The problem with it becomes obvious after running the system some time. The predictions keep getting worse, making an increasing amount of predictions unreliable, and the robot relies more and more on on random angles. Given enough time, the robot will get the block to the goal position by randomly choosing angles (Pólya, 1921), however, that would not be of any help, since the the robot has learned nothing, and will not be able to use this to improve.

In figure 4, some regions have unnecessarily many points remembered, which is one symptom of accumulating noise. A few really bad outliers can also be seen. There exist several methods for pruning, and several more might still be undiscovered, however, none have been tried in this project. One large problem when pruning is how to separate genuine errors from other phenomena, such as a new shape appearing. Since the robot is learning about the world from scratch, it can be hard to even distinguish between a good and a bad point altogether.

Despite these problems, the robot can learn how to perform targeted pushing of objects. It does this, relying on a minimal amount of planning, which separates it from approaches in classic AI. A feature that separates Radagast from more recent AI-approaches, is that it will learn continuously while it is running. When deciding whether or not the latest experience should be remembered or not is not whether it was successful or not, but rather whether the observed behaviour was predictable or not. This method will, as mentioned before, lead to over training if no pruning function is implemented, since strange and genuinely wrong behaviours should be unpredictable.

Fitzpatrick et al. (2003) let a robot handle a similar problem as the one in this case, however, they only let the robot learn during a learning phase. During the learning phase, several predetermined actions, such as pulling and slapping, are tested several times, to make lists on the relation between the actions and reaction for different objects. Stoytchev (2005), who train robots to use tools, also use initial learning trial, where all the predetermined behaviours are tried with all the tools, to get a complete list of action-reaction-relations. Only learning in a specific and initial phase will make the risk of over training decrease. This is a good feature, which is needed in this project, however, it will not be able to handle novel situations. That would be a big problem for Radagast, so a different solution (such as good pruning) is preferred over introducing learning phases. It might be possible to introduce conditions to engage a learning mode, so that the robot realises that it need to learn something new. This is similar to what is intended to be implemented (and partly is), however, introduction of pruning would probably still help, so the only addition would be that an end condition would be necessary to end the learning phase.

Another important use for pruning is to manage the prototypical shape, which starts as a copy of the first shape, and then evolves by adding new points, which are common to several shapes. Apart from accumulating noise in the same way as the other shapes, it was created with noise, since the first shape probably has some shape specific points, which the prototype inherits. These points needs to be pruned away in the prototype, which could be done over time, as the systems learns about more shapes.

As mentioned before, this robot have only operated in a plane. This is a simplification compared to the three dimensional world in which we live. To make a robot perform an equivalent task in three dimensions would not add much conceptually new to the problem, but the problems when implementing would increase much. If the plane in which the objects move were kept as it is, but the objects were exchanged for three dimensional counterparts, more easily distinguishable push reactions might occur. Flat objects might for example slide, while round objects might roll. If the robot is not fast enough, or is not good enough to handle its body, objects able to roll

might easy roll out of reach, which would be troublesome.

One source of noise is the inaccuracy of the module detecting the orientation of the blocks. To improve this, several more rotated versions of the original image of each shape is needed, which would improve the general performance of the module. The downside to this is that the time it takes to execute the module increases significantly for each extra image, however, due to symmetries in the shapes, some orientations are indistinguishable similar, which can be used to reduce the number of needed images. This issue is not so pressing since other kinds of noise will occur, so a more general method of removing noise is prioritised.

A good feature that the ShapeMod module has, is that it often can detect the correct shape, and its location, even if it is partially hidden. It is common that the arm or the finger hide parts of the shape from the camera, when moving around, which make this feature very useful. It has, however, its limits, and one way to push these limits is by utilising the depth detection feature of the Kinect. This could help the shape detection module, by providing information on whether there is something between the object and the camera or not. It has not yet been implemented, mostly because it has not been necessary, but it should not be to hard to do. One concern might be that the interesting objects to look at is out of range of the depth detection of the Kinect, however, it should be possible to find ways around this if the feature is needed.

The most important things to improve, to continue on this work, is to add pruning of the learned data, and to switch to some other representation of the angles, to avoid the discontinuity. As mentioned before, pruning is a common problem, with several solutions, however, it is necessary to choose and adapt the algorithm carefully and specifically for the problem. One solution to the problem with the angles are, as mentioned before, to use trigonometric functions, which have the drawback that the problem increases in size, and the system needs more time to run. When these things are sorted out, more advanced features (with its own set of problems that will appear when tested) have been prepared. For example, given that a good condition is found, it should be possible for the robot to recognise and handle new shapes as they appear. It should be able to do this, without starting from scratch with each shape, since one level of prototype shape is implemented. Abstract prototypes for shapes could be considered understanding of affordance of pushable objects, and the knowledge of the more specific shapes could be considered understanding of affordance of objects with more specific features. This should be achievable without knowing what the objects are, that is, without any semantic knowledge.

Using linear splines as a fast way to learn general behaviour from few examples seems to be effective if the function needed to be learned is well-behaved. To fast realise the general use of objects, given only sparse visual information, is what happens when understanding of affordances is used along the how stream. Similar behaviour has been seen in Radagast, however, the choice of inner representations introduces an unwanted and non-physical restriction. The usefulness of the well-behaving parts will decay over time, due to the accumulation of noise. Solutions to both these problems have been suggested, so a continuation of the project should be possible and interesting.

## Acknowledgements

I would like to thank my supervisor Christian Balkenius for all the guidance and constructive input during this project. I would also like to thank Rasmus Bååth and Birger Johansson for all the help and support that they have provided.

## References

Balkenius, C., & Johansson, B. (2007). Finding colored objects in a scene. *LUCS Minor*(12).

Çakmak, M., Doğar, M., Uğur, E., & Şahin, E. (2007). Affordances as a framework for robot control. In *Proceedings of the 7th international conference on epigenetic robotics, epirob'07.*

Şahin, E., Çakmak, M., Doğar, M. R., Uğur, E., & Üçoluk, G. (2007). To Afford or Not to Afford: A New Formalization of Affordances Toward Affordance-Based Robot Control. *Adaptive Behavior*, *15*(4), 447–472.

Fitzpatrick, P., Metta, G., Natale, L., Rao, S., & Sandini, G. (2003). Learning about objects through action-initial steps towards artificial cognition. *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, 3140–3145.

Gibson, J. J. (1977). Perceiving, acting, and knowing: Toward an ecological psychology. In R. Shaw & J. Bransford (Eds.), (chap. The Theory of Affordance). Michigan: Lawrence Erlbaum Associates.

Goodale, M. A., & Milner, A. D. (1992). Separate visual pathways for perception and action. *Trends in neurosciences*, *15*(1), 20–5.

Hodges, J. R., Bozeat, S., Lambon Ralph, M. A., Patterson, K., & Spatt, J. (2000). The role of conceptual knowledge in object use evidence from semantic dementia. *Brain : a journal of neurology*, *123 ( Pt 9)*, 1913–25.

Kirsh, D., & Maglio, P. (1994). On Distinguishing Epistemic from Pragmatic Action. *Cognitive Science*, *18*(4), 513–549.

Lauro-Grotto, R., Piccini, C., & Shallice, T. (1997). Modality-specific operations in semantic dementia. *Cortex; a journal devoted to the study of the nervous system and behavior*, *33*(4), 593–622.

MacDorman, K. F. (2000). Responding to affordances: Learning and projecting a sensorimotor mapping. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, *4*, 3253–3259.

Pólya, G. (1921). Über eine aufgabe der wahrscheinlichkeitsrechnung betreffend die irrfahrt im straßennetz. *Mathematische Annalen*, *8*(1), 149–160.

Rossetti, Y., Revol, P., McIntosh, R., Pisella, L., Rode, G., Danckert, J., et al. (2005). Visually guided reaching: bilateral posterior parietal lesions cause a switch from fast visuomotor to slow cognitive control. *Neuropsychologia*, *43*(2), 162–177.

Stoytchev, A. (2005). Behavior-Grounded Representation

of Tool Affordances. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 3060–3065.

Uexküll, J. von. (1957). Instinctive behavior: The development of a modern concept. In C. E. Shiller (Ed.), (chap. A stroll through the worlds of animals and men: A picture book of invisible worlds). New York: International Universities Press, Inc.

Ungerleider, L. G., & Mishkin, M. (1982). Analysis of visual behavior. In D. J. Ingle, M. A. Goodale, & R. J. W. Mansfield (Eds.), (chap. Two Cortical Visual Systems). Cambridge: The MIT Press.

Uğur, E., Şahin, E., & Öztop, E. (2009). Affordance learning from range data for multi-step planning. In *Proceedings of the 9th international conference on epigenetic robotics, epirob'09.*