Learning from Demonstration: An Investigation into the use of Predictive Sequence Learning (PSL) for Robot Manipulation

Final Research Project (04-990)
MSIT
Victor Akinwande
Carnegie Mellon University Africa

May 21, 2018

Abstract

Robot learning from demonstration is a paradigm in robotics that involves a robot learning behaviors from a set of examples and provides a more intuitive way to teach new skills to robots in environments they had not been programmed to act. This work focuses on developing a learning technique or action policy for a robot manipulation task involving picking and placing an object using teleoperation based on algorithm known as Predictive Sequence Learning (PSL). PSL has previously only been applied to navigation tasks on a mobile robot and this report investigates the use of PSL as a controller for a 5 degree-of-freedom robot arm performing a manipulation task.

1 Introduction

Robot programming is typically a complex process involving explicit detailed specification of the position and orientation of the robot as well as the objects it is manipulating. This poses a hindrance to non-robotics experts that will increasingly be presented with opportunities to interact with robots in the society. Programming by demonstration (PbD) also known as learning from demonstration (LfD) is an approach to learning behaviors from a set of examples called demonstrations. LfD presents a more intuitive way to learn new skills for robots and involves perception and analysis of the demonstration, representation of the demonstration and reproduction of the task on the robot [13]. Another motivation for LfD stems from the fact that it is not feasible to take into account all the possible environments in which a robot would be put to work during design and task specification may vary. As such, the ability for a robot to pick up new skills easily is one that is much desired. LfD is thus geared to open up the ability to control

the actions of robots to non-robotics experts and develop a more intuitive medium for communication from humans. This is important as robots are set to become more common in everyday settings [3].

In LfD, there are two parties: the Robot (learner) and the Demonstrator (teacher) which is but not necessarily human. In general, there are two phases in a LfD problem: gathering the demonstrations and deriving a mapping between the world state and an action(s) to be taken by a robot. In addition, for LfD to be successful the state(s) and action(s) pairs of the teacher need to be available for use by the robot. We are however faced with two main problems: (i) the possible difference in the morphology of the robot and the teacher and (ii) the need for the robot to access the demonstration by the teacher while accurately capturing the demonstrated behavior. In effect, there has to be some mechanism for the robot to record the actions of the teacher in spite of the morphological difference, and a mechanism to map or determine the correspondence of this recording to the embodiment of the robot. These two mechanisms are known as *record* and embodiment mapping respectively. Furthermore, either of the two mechanisms can be direct or indirect. If it is direct, then the data can be used by the learner without modification. If it is indirect, the data has to be mapped to the robot.

As such, these mappings give rise to four general approaches to record and embody demonstrations: (i) *Teleoperation* - the robot is operated by the teacher while recording from its own sensors e.g. flying a drone to teach it how to fly; (ii) *Shadowing* - the robot mimics the teachers demonstrations while recording from its own sensors e.g. a humanoid learning arm gestures; (iii) *Sensor on teacher* - an imitation technique in which sensors on the teacher are used to record the demonstration e.g using sensor suits on a human to record joint angles as they teach drumming patterns to a humanoid; and (iv) *External observation* - where sensors external to the teacher are used to record its demonstrations. e.g. a robotic arm learning pole balancing via stereo vision to track the movement of the pole while it is being balanced by a person.

Finally, there is a third mapping called the action-policy mapping which arises due to the need to map the sensor-state of the robot or its environment to corresponding actions. This could be addressed by either approximating a mapping function based on the demonstration data, or using the data to create a model of the world dynamics, or learning rules associated with a set of pre-and post-conditions applicable to the robot [3]. This taxonomy of LfD comes from [3].

This research project uses teleoperation LfD for a 5 degree-of-freedom manipulator robot performing a pick and place task and employs the predictive sequence learning algorithm (PSL) as the sensor state-action mapping technique. Precisely, the robot's joints are controlled in a 4 degree-of-freedom Cartesian frame of reference using a 6D joystick. The choice of

teleoperation LfD removes the need to develop complex record and embodiment mapping strategies as the robot is directly controlled by the teacher and its sensors directly embody the behaviours. PSL has only been applied to mobile robot navigation, and our goal is to investigate how it can be adapted for manipulation tasks. The use of PSL, however, simplifies the exercise by focusing on discrete rather than continuous movement while still providing a way of combining learned policies using internal simulation to effect novel manipulation strategies.

2 Literature survey

The previous section is heavily based on the LfD taxonomy proposed in [3] and widely accepted as a source that presents an overarching categorization of the various LfD approaches. Another well known taxonomy is presented in [4]. Similar to [3], their work provides a review of LfD approaches in the two phases which [3] highlights as gathering a demonstration, and deriving a mapping between the world state and an action(s) but is referred to as determining *what to imitate* i.e encoding a skill and *how to imitate* learning a skill respectively. This section places the reviewed literature with the framework of these two taxonomies.

Motivated by the lack of a unified formalism for the different components that constitutes research in LfD, [10] presents a definition and formalization of the the common ideas and principles involved in LFD. According to the formalism, an agents sensory-motor history is described by an event history, and a controller maps event histories to actions in action space. A demonstration of a behavior is seen as an event history and the behavior itself as the large set of allowed event histories, i.e., all possible ways to realize the desired behavior. The quality of the learned controller can be judged by the similarity between the set of allowed event history and the realization space: the set of event histories generated using the learned controller. In addition, the meaning of repeating a demonstrated behavior is discussed from a machine learning perspective and the concept of generalization is defined in the framework of event histories and leads to a discussion of bias in learning. Slight variations to the execution of a task is referred to as bias and is essential in LfD, and can be introduced before, during, and after demonstration as feedback from the teacher.

The formalism presented in [10] is heavily focused on the concepts surrounding learning a skill or deriving an action policy and LfD is even described simply as controller selection involving three steps:

1. Behavior segmentation: dividing the demonstration into smaller segments

2. Behavior recognition: associating each segment with a primitive and

3. Behavior coordination: identifying rules or switching conditions for how the primitives are to be combined.

2.1 Representation of skills

It can be argued that the method for gathering a demonstration or encoding a skill is equally as crucial as the method for enabling the robot to learn the encoded skill.

On the subject of encoding skills, there are several approaches that have been employed in mobile robotics and manipulation. In [13] for example, the authors argue that the representation of manipulation knowledge in LfD can be subsymbolic or symbolic and present a subsymbolic approach based on learning Dynamic Motor Primitives (DMP), and also a partially symbolic approach, based on learning the parameterization of predefined movement primitives: meaningful elements that are combined to form actions. Similarly, and demonstrated on manipulation tasks, [1] propose using specific robot configurations marked by the teacher called keyframes as opposed to complete trajectories. These configurations are then stored as a sequential set of discrete end-effector configurations.

The authors of [5] develop a model composed of a hierarchical time delay neural network that extracts invariants from demonstrated manipulation tasks. The system analyzes the trajectories of both the objects involved and the arms of the demonstrator and uses the probability of event occurrence to determine the goals of the manipulation task. Certain manipulation tasks are force based, and in [18], the authors propose using force perception through haptic feedback as a means for improving demonstrations inspired by the assumption that force conveys relevant information in the absence of vision or motion sensing. They describe a technique based on the mutual information between perceptions and actions and encode demonstrations using a Hidden Markov Model (HMM).

2.2 Suboptimal demonstrations

Suboptimality in a demonstration during LfD could exist due to the demonstrator demonstrating unnecessary, incorrect, or unmotivated actions, or the ambiguity presented with choosing a scenario to apply an action, and where the actions are demonstrated with the wrong intention. In [12], the authors describe strategies for dealing with suboptimality by posing the optimality problem as a noise removal problem and propose a two-step approach for coping with demonstration suboptimality: (1) determine the geometric properties of the task from demonstration, and (2) determine the optimized robot control commands based on the information obtained in (1). The authors go on to derive an explicit and quantitative geometric representation of the task providing a source of information for building strategies to cope

with demonstration suboptimality by using statistical regression analysis on demonstrated paths to derive a geometric representation of a task that focuses on the constraints on motion caused by one object in the task on another, rather than on the dimensions of the objects themselves.

2.3 Deriving an action policy

We will now address the question of learning the appropriate policy to map between the world state and the action to be taken by the robot. [17] develop a generalized policy from the trajectories (sequence of positions) of the end-effector of a 6-degree-of-freedom robot arm in the demonstration stage by reducing those trajectories to their mean and standard deviation. The trajectories are point-wise defined and gained by the robot by calculating relative trajectories of objects to the initial coordinates of objects involved and the distance between them and the end-effector. Areas where the deviation is small, indicate strong compliance during all demonstration and are regarded as constraints. During reproduction of learned behavior, the coordinates of the objects are used as an offset for the averaged trajectories.

Conversely, [14] propose a technique for learning the action policy mapping for LFD called constructing skill trees (CST). Rather than converting a demonstration trajectory into a single controller, CST segments demonstration trajectories into a sequence of controllers or behaviors termed skills. CST extracts skills which have goals in particular, and the objective of skill n is to reach a configuration where skill $n + 1$ can be successfully executed. Given a cost function, they use a reinforcement learning based policy learning algorithm to improve the robots performance for each individual skill.

In like manner, the authors of [2] extend this idea of encoding individual skill by incorporating simpler behaviors learned from demonstration into larger tasks. They introduce a technique that uses corrective human feedback to build a policy able to perform an un-demonstrated task from simpler policies learned from demonstration. Teacher feedback is then used to enable and improve policy behavior at transition points that link demonstrated primitives by continuous-valued corrections on student executions as an alternative to providing further demonstrations. However, new example state-action mappings are synthesized, not from teacher demonstration but from teacher feedback and learner executions, without requiring state re-visitation by the teacher to provide appropriate behavior information. In the paper, an action policy is learned by function approximation being performed via regression given that the target application involving low-level motion control had a continuous action space.

2.4 Performance of demonstration

Gathering demonstrations can be tedious particularly with approaches such as teleoperation. Increasingly there has been a shift towards self-supervised gathering, and external observation of demonstrations by the robot. In [19], the authors propose a technique to learn representations of demonstrations from third person video observations. They also extend this technique to directly imitate human pose. This representation known as time-contrastive embedding (TC) is trained by minimizing a triplet loss defined by temporally co-occurring frames (frames from the same time with different viewpoints) and negative frames formulated to ensure that co-occurring frames are closer in an embedding (high dimensional vector representation) space than negative frames. To mimic human demonstrations and poses without any joint level alignment, the TC network is trained on a diverse set of demonstrations with different embodiments, objects, and backgrounds. The TC embedding is then used to create a reward function for reinforcement learning of robotic manipulation skills.

The idea of self-supervision was extended to learning rope manipulation skills in [15]. The authors train a predictive convolutional neural network model of rope behavior using data of rope interaction collected autonomously by the robot. This model predicts, given the current and target image of a rope, which action the robot can execute to put the rope into the target configuration. In LfD this is referred to as an inverse model. To learn a particular skill, a teacher supplies step-by-step images that show the rope undergoing a variety of manipulations. The inverse model is then used to predict the sequence of actions needed to reproduce the tasks during execution.

2.5 PSL

Finally, PSL is a model free learning algorithm that is able to perform the action-policy mapping for a LfD task by storing up knowledge as sequences of sensory-motor relations called *hypothesis*. PSL is used in this work as a skill learning algorithm but can also be used as a behavior recognition technique i.e to predict the sensory consequences of a motor command [11]. The PSL inventors also go on to demonstrate that this is sufficient to learn robot navigation skills by a mobile robot [9]. However, the modifications to achieve this involved introducing *fuzzy* rules to describe the temporal relations between sensory-motor events recorded during demonstration. The specific version of PSL used is describes in the following section.

Algorithm 1 Predictive Sequence Learning

Require: an event sequence $\eta = (e_1, e_2, \ldots, e_n)$

1: $t \leftarrow 1$
2: $H \leftarrow \emptyset$
3: $M \leftarrow \{h \in H | X_h = (e_{t-|h|+1}, e_{t-|h|+2}, \ldots, e_t)\}$
4: **if** $M = \emptyset$ **then**
5: **let** $h_{new} : (e_t) \Rightarrow e_{t+1}$
6: **add** h_{new} to H
7: **goto** 21
8: **end if**
9: $\hat{M} \leftarrow \{h \in M |\ |h| \geq |h'|$ *for all* $h' \in M\}$
10: **let** $h_{max} \in \{h \in \hat{M} |\ c(h) \geq c(h')$ *for all* $h' \in \hat{M}\}$
11: **if** $e_{t+1} \neq I_{h_{max}}$ **then**
12: **let** h_c be the longest hypothesis $\{h \in M | I_h = e_{t+1}\}$
13: **if** $h_c = null$ **then**
14: **let** $h_{new} : (e_t) \Rightarrow e_{t+1}$
15: **else**
16: **let** $h_{new} : (e_{t-|h_c|}, e_{t-|h_c|+1}, \ldots, e_t) \Rightarrow e_{t+1}$
17: **end if**
18: **add** h_{new} to H
19: **end if**
20: update the confidences for h_{max} and h_c
21: $t \leftarrow t + 1$
22: **if** $t < n$ **then**
23: **goto** 3
24: **end if**

Algorithm 2 Making predictions using PSL

Require: an event sequence $\eta = (e_1, e_2, \ldots, e_t + 1)$
Require: the trained hypotheses library $H = (h_1, h_2, \ldots, h_{|H|})$

1: $M \leftarrow \{h \in H | X_h = (e_{t-|h|}, e_{t-|h|+1}, \ldots, e_{t-1})\}$
2: $\hat{M} \leftarrow \{h \in M |\ |h| \geq |h'|$ *for all* $h' \in M\}$
3: **let** $h_{max} \in \{h \in \hat{M} |\ c(h) \geq c(h')$ *for all* $h' \in \hat{M}\}$
4: **return** the prediction $\hat{e}_t = I_{h_{max}}$

3 Predictive Sequence Learning (PSL)

Predictive sequence learning (PSL) [6] is a sequence learning algorithm that was designed to be applied as a technique for "learning a skill" in a learning from demonstration framework. It is capable of being used as a sensor state-action mapping technique and has been applied to mobile robot navigation in both simulated and real environments.

PSL has various formulations and forms. Discrete PSL, discussed in this section, Fuzzy PSL to cater for where the state space has many dimensions and the data for each dimension is continuous, and lastly Covert PSL to enable the use of PSL in covert mode as a predictor of an observation (forward model). The scope of this work is limited to the use of discrete PSL (DPSL), it should thus be noted that further references to PSL in this text refer to DPSL.

Given a sensory-motor space $I = U \times Y$, where U and Y denotes the action and observation space respectively, PSL extends the sensory-motor space by introducing temporal structure to that formulation. A data structure known as an event sequence η is built up by alternating action and sensor/observation events during demonstration i.e $\eta = (u_1, y_1, u_2, y_2, ..., u_t, y_t)$

PSL learns by prediction and is trained on an event sequence $\eta = (e_1, e_2, \ldots, e_t)$ that is defined up to a time t from where the next event is to be predicted. Knowledge is represented as a set of hypotheses, called the hypothesis library H. a hypothesis $h \in H$ denotes a dependence between an event sequence $X = (e_{t-|h|+1}, e_{t-|h|+2}, \ldots, e_t)$ and a target event $I = e_{t+1}$:

$$h : X \Rightarrow I \tag{1}$$

We call I the head and X the body of the hypothesis.

Every hypothesis h is associated with a metric that reflects the probability of an event sequence X given a target event: $P(I|X)$ called *confidence* c. For a given η, the confidence metric is based on another metric that describes the proportion of sub-sequences, the same size as X, in η and called support s.

In other words, $c(X \Rightarrow I) = s(X, I)/s(X)$ where (X, I) denotes a concatenation of X and I.

PSL can be used to make predictions based on the sequence of past events or observed event sequence η. In the learning phase, the algorithm begins with an empty hypothesis library H and tries to predict a future event e_{t+1} based on η. If it fails, a new hypothesis h_{new} is added to H i.e a dependence between η and the correct event e_{t+1}. As a result, PSL learns only when it fails to correctly predict a future event. This algorithm is shown in Algorithm 1.

To thus make a prediction during execution, given an event sequence,

PSL selects the longest matching hypothesis body with the highest confidence score and returns the head of that hypothesis as its prediction for the future event. Matching in the original implementation of PSL involves a perfect correspondence containment or simply equality.

To recap:

Y is the observation space	H is a set of hypotheses
U is the action space	$X = (e_{t-\|h\|+1}, e_{t-\|h\|+2}, \ldots, e_t)$
$I = U \times Y$ is the sensorimotor space	X_h is the body of the hypothesis
e is an event, $e \in \sigma$ where $\sigma = U \bigcup Y$	I_h is the head of the hypothesis
$\eta = (e_1, e_2, \ldots, e_t)$	$c(X \Rightarrow I) = s(X, I)/s(X)$ is the confidence score on a hypothesis
$\eta = (u_1, y_1, u_2, y_2 \ldots, u_t, y_t)$ such that events are drawn alternately from U and Y	$s(X)$ is the support of X, i.e the number of instances of X in an event sequence from the time of creating that hypothesis to $t-1$
e_{t+1} is an event to be predicted	(X, I) is a concatenation of X and I
$I = e_{t+1}$ is a predicted event	$s(X, I)$ is the support of (X, I) i.e the number of instances of (X, I) in an event sequence from the time of creating that hypothesis to t
$X = (e_1, e_2, \ldots, e_t)$ is a preceding event sequence	
$h \in H$ and $h : X \Rightarrow I$ is an hypothesis capturing dependence between a given X and I	

Learning generates a hypotheses library that is effectively the action policy.

Execution involves applying the action policy based on real-time observation and executed actions selected through predictions as shown in Algorithm 2.

3.1 Calculating hypotheses confidence scores

A Javascript implementation of PSL is open-sourced by the authors of the algorithm on GitHub, and this implementation was ported into C++. It is noteworthy that the formulation for calculating confidence scores for the hypotheses as proposed in the original paper, [6], varies in this implementation. Whereas the paper employs the concept of support i.e $s(X)$ to calculate confidence scores, the implementation uses constructs called *hits*

and *misses*. In other words, a hypothesis has these two attributes: *hits* and *misses* that are updated during training. A *reward* function increments *hits* by 1 when a hypothesis correctly predicts I and increments *misses* by 1 when it incorrectly predicts I.

Thus, as usual, given an event sequence $\eta = (e_1, e_2, \ldots, e_t)$, the hypothesis library is optionally updated using the information at every time-step. The confidence score for every hypothesis, referenced by its index: $scores[i]$ and given the current event sub-sequence at timestep j: $sub = \eta[1, j]$, is to calculated to be:

$$scores[index] = \begin{cases} (hits/(hits + misses)), & \text{if sub } <=> X_h \\ & \text{or sub } <=> X[-|sub|] \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Where:

$hits$ is initialized to 1,

$misses$ is initialized to 0,

X_h is the body of the hypothesis

$|sub|$ is the length is the size of the sub-sequence

$<=>$ represents some form of matching.

In the original implementation, $<=>$ implies an exact matching of the sequences i.e $X = (e_1, e_2, \ldots, e_m)$ matches $\eta = (e'_1, e'_2, \ldots, e'_n)$, if and only if, for some integer $k, e_i = e'_{i+k}, \forall i \ldots m$

3.2 Event sequence matching

In PSL, events are matched using equality indicative of an exact matching function as earlier discussed. While this may have proven to work well with mobile robot navigation, for a manipulation tasks and particularly with two sensing sources that potentially introduce noise to the demonstration an exact matching may not be desired and may lead to the inability to afford small variations in the execution of a task. In other words, since the hypothesis library remains fixed after demonstration, if a situation is encontered at execution that was not encountered during demonstration, no action can be predicted and execution ends. To address this, we propose the use of an approximate matching algorithm: the Levenshtein distance algorithm and compare its performance with the exact matching in PSL. The Levenshtein distance algorithm gives the minimum edits required to transform one string to another. Thus, instead of using an exact matching function, a threshold

n is set to be the maximum edits for two event sequences to match. In other words, the match function could potentially match ${}^{(5*|\eta|)}P_n + 1$ event sequences for a given n, and taking into consideration that both the action and observation events are 5-sized tuples. This should introduce robustness to the learned hypothesis library and limit the need for all potential situations in the execution of a task to be present in the demonstration.

Mathematically, the Levenshtein distance between two string a, b of length $|a|and|b|$ respectively is given by $\text{lev}_{a,b}(|a|, |b|)$ where:

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if}\min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1,j) + 1 \\ \text{lev}_{a,b}(i,j-1) + 1 \\ \text{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{3}$$

4 Adaptation of PSL to robot manipulation: a pick and place task

PSL operates in two phases, the learning or training phase where a library of hypotheses is derived from demonstrations and the execution phase, where the library is used to control the robot by continually predicting the next action to take based on the sequence of sensory-motor events.
To adapt PSL we need to identify the following:

- The observation or sensor space Y.
- The motor command or action space U.
- The match function for events sequences.
- The hypotheses matching function

As we would like to make the robot manipulation friendlier to non-robotics experts, it makes sense to use a Cartesian frame of reference as opposed to joint positions (specified as servo pulse width values). The inverse kinematics solution gives us this ability. In the pick and place application, objects are grasped from above with a two-finger end-effector. With the z axis of the end-effector facing downwards, 3 representational degrees of freedom are determined by the position of the robot wrist. Rotating about the z axis constitutes another degree of freedom and is determined by the orientation. We also control the gripper which could be *open or close* Thus, the action space can be specified using a 5 dimension sized tuple: x, y, z, θ, g.

As we also wish to constrain the size of the action space and ensure it is adaptive and invariant to fixed positions it is preferable to adopt a

differential action space. Thus we have our action space to be Δx, Δy, Δz, $\Delta\theta$, each dimension quantized to constrain its size, and a binary gripper variable g.

Hence, we have u_t as follows.

$$u = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\theta \\ g \end{bmatrix} \tag{4}$$

The observation space, Y has nine parameters, four for the end-effector E_x, E_y, E_z, and E_θ, four for the object to be manipulated O_x, O_y, O_z, and O_θ and one for the gripper state g to keep track of the last known binary state of the gripper and determined from the previous action. Recall that the first event in the event sequence of the PSL formulation is an action. Similar to the action space, we choose a differential observation space defined as follows.

$$y = \begin{bmatrix} O_x - E_x \\ O_y - E_y \\ O_z - E_z \\ O_\theta - E_\theta \\ g \end{bmatrix} \tag{5}$$

This mirrors the observation state space in PSL in [6, 7] which is based on the distance between the robot and objects in its environment. In other words, the tuples in Eqn. (4) and Eqn. (5) represent the action and observation events respectively in our sensory-motor space I.

5 Lynxmotion AL5D robotic arm

The AL5D robot arm has 6 motors providing 5 degrees of freedom and 1 gripper control. The manipulator movements are programmed by specifying the joint positions. The *forward kinematics* of the manipulator allows us to compute the position and orientation of the end-effector in a 3D Cartesian frame of reference given the manipulator joint positions. The *inverse kinematics* allows us to compute the corresponding joint positions given a desired position and orientation (pose) of the end-effector ($T6$) [16][20].

It is noteworthy that $T6$ is the function of the joint variables for a full 6 degree-of-freedom pose. However, we only have 5 degrees-of-freedom and need 4 degrees-of-freedom since the end-effector is always facing downwards in our pick and place task.

We use homogeneous transformations to represent the spatial relationships between the manipulator and objects in its environments. A homo-

Figure 1: Lynxmotion AL5D Robotic Arm with a BotBoarduino microcontroller

geneous transformation represented by a matrix describes the position and orientation (pose) of a co-ordinate frame with respect to a previously defined frame (base), and also how the base frame was transformed to the new frame [21]. The end-effector of the robot arm can be described by two transformations: the transformations leading from the base to the wrist to the end-effector: $Z *^{Z} T6 *^{T6} E$ and by the transformations leading from the block to the end-effector grip position: $B *^{B} G$.
Where:

Z - is the transform which describes the position of manipulator with respect to the base co-ordinate reference frame

${}^{Z}T6$ - describes the end of the manipulator (i.e. the wrist) with respect to the base of manipulator, i.e. with respect to Z

${}^{T6}E$ - describes the end-effector with respect to the end of the manipulator, i.e. with respect to T6

B - describes a blocks position with respect to the base co-ordinate reference frame

${}^{B}G$ - describes the manipulator end-effector with respect to the block, i.e. with respect to B.

By equating these two transformations, and solving for $T6$ we obtain a function of the joint variables of the manipulator and, if known, the appropriate joint variables can be computed using the inverse kinematic (IK) solution. We have the inverse kinematic solution and thus we are able to compute the corresponding joint position given a pose of the end-effector.

6 Implementation

6.1 SSC-32 controller

All experiments are carried out on a robot arm which is the Lynxmotion AL5D robot arm previously described. The SSC-32 controller serves a servo controller providing 32 channels of 1μs resolution servo control. The SSC-32 works with pulse proportional servos. The signal it transmits and receives consists of positive going pulses ranging from 0.9 to 2.1ms (milliseconds) long, repeated 50 times a second(every 20ms). The servo positions its output shaft in proportion to the width of the pulse. The positioning resolution is documented to be 0.09°/unit (180°/2000), but slight variations are observed in the servos and they were all calibrated individually to match the joint angles with the corresponding pulse width. Some servos may not be able to move the entire 180°range and this is documented for the AL5D robot arm.

The "servo movement" command group controls movement of a servo or multiple servos by specifying the pulse width. It uses the syntax as described below.

$\#< ch > P < pw > S < spd > ...\#< ch > P < pw > S < spd > T < time >< cr >$

- $< ch >$=Channel number in decimal, 0-31
- $< pw >$=Pulse width in microseconds, 500-2500
- $< spd >$=Movement speed in μs per second for one channel (optional)
- $< time >$=Time in msec for the entire move, affects all channels, 65535max (optional)
- $< cr >$=Carriage return character, ASCII13 (Required to initiate action)
- $< esc >$=Cancel the current command, ASCII27

ServoMoveExample:"#5P1600S750$< cr >$"

The example will move the servo on channel 5 in proportion to the pulse width of size 1600. It will move from its current position at a rate of 750μs per second until it reaches its commanded destination. Consider that 1000μs of travel will result in around 90°of rotation. A speed value of 100μs per second means the servo will take 10 seconds to move 90°. Alternately a speed value of 2000μs per second equates to 500ms to move 90°.

6.2 The manipulation task

We define the pick and place task to be picking up an object, specifically a paper highlighter that lies vertically in the direction of the y-axis of the

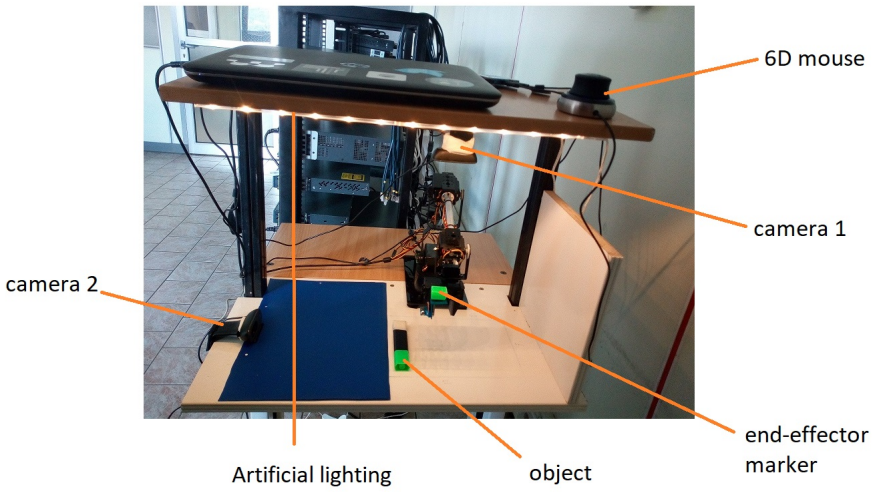

Figure 2: Experimental setup showing the robot arm, the sensors and the joystick

manipulator wrist. The object is then lifted for a few centimeters in the z-axis. Vision determines the specific pose of the object by *segmenting* the cover of the highlighter where the body of the highlighter serves as an offset to the end-effector (a two finger gripper).

6.3 Setup

Given the manipulation task, the object is grasped from above using the gripper. We mount two cameras on the work surface. One facing vertically downwards and the other facing horizontally sidewards and mounted coplanar to the initial position of the object. A marker is placed on a small offset attached to, and parallel to the base of the end-effector. The marker is clearly visible by both cameras and gives an estimate of the pose of the end-effector. Based on this setup, the downward camera yields the x and y positional components as well as the angle and the sideward facing camera yields the z positional component of the observation space.

Artificial lighting is placed above and beside the work surface to compensate for the variability of natural lighting. The robot arm is also fitted to the work surface using velcro tapes. This setup is shown in Figure 2.

6.4 Object tracking

To determine the differential pose of the object based on the formulation discussed earlier, the object and the end-effector marker are segmented in hue and saturation color spaces from the image frames gotten from both

cameras. Hue and saturation values of a color vary as lighting varies, even slightly. To deal with this, an interval of hue and saturation values estimated using images of the same objects under different lighting conditions form the basis for segmentation during demonstration and execution. This is aided by using the Open CV library. The difference of the object and end-effector pose in image space yields the differential pose and the resulting values are quantized to $[-10, 10]$ or $[-20, 20]$ selected heuristically in an attempt to reduce the observation space.

6.5 6D mouse integration and control

Demonstration data is gathered using teleoperation. To enable this, we employ a 6D mouse that allows for control in the x, y, z, Cartesian axes. Specifically, the 3Dconnexion Space Navigator Mouse is used for this purpose. The mouse provides 6 discrete values: x, y, z and rotation about the x, y, and z axes in the interval $[-350, 350]$, but we we only use 4 of these dimensions. An interface is developed for the mouse that maps its frame of reference to that of the wrist of the robot arm. The orientation is mapped to be the rotation about the y-axis on the mouse. To enable control of the robot arm however, the mouse is not used to provide positional values but rather to provide deltas for each axes. In other words, once a motion event is detected by the mouse interface, the axis with the highest absolute magnitude is selected as the axis of control. A delta is then added to the current position of the robot arm in that axis. The new position is sent to an interface developed for the SSC-32 controller that maps the Cartesian poses to the corresponding joint positions and executes this on the controller. Finally, the left button of the mouse controls the binary gripper state of the robot. In this manner, control is enabled using the 6D mouse.

6.6 Demonstration and execution

To recap, both the action and observation space are quantized. The action space is quantized to fixed delta for each of the $\Delta x, \Delta y, \Delta z, \Delta\theta$ values and the observation space is quantized into $[-10, 10]$ or $[-20, 20]$ for each of the differential x, y, z, θ values depending on the experiment being performed. This yields an action space of size ${}^5C_2$ i.e including the gripper action and an observation space of size 20^5 or 40^5. Quantizing the spaces constitutes some form of design bias which aids learning [10]. The impact of these quantizations on the action policy are characterized in the following section.

The *approximate matching threshold* also constitutes some form of design bias. Recall that for a given threshold n, the match function could potentially match ${}^{(5*|\eta|)}P_n + 1$ event sequences. This is called the *match window*. The only way to address the eventuality of matching incorrect event sequences during execution is through repetition, effectively increas-

ing the confidence of the repertoire of hypotheses that actually capture the demonstrated task. One of the common sources of errors during demonstration arose from the fact that due to the sensitivity of the 6D mouse controlling the z axis often results in control of the rotation about the y axis which is mapped to the orientation of the end-effector.

Demonstration stage frequency is defined as the frequency to which demonstrations are recorded. This could be varying - based on mouse events i.e when an action is taken or constant where the observation is continually being recorded and the action is the most recent action since the last timestep. If no motion event was triggered by the mouse interface, an action event with *no action* is recorded. Demonstrations are written to a text file as alternating action and observation events. During execution, an event sequence denoted η is built up equally by alternating action and observations events at every stage k using a stage frequency of $2Hz$, producing one observation and action every 0.5 seconds. Since the last element of η is an observation, PSL will predict an action and the preceding observation is added to the event sequence, transforming stage k to $k+1$.

Here is an example of data based on the formulation in Eqn. 4 and Eqn. 5 represented as an alternating sequence of action and observation events: { ... (0 0 3 0 0), (-1 -3 6 0 0), (0 0 3 0 0), (-1 -3 5 0 0), (0 0 3 0 0), (-1 -3 5 0 0) }.

7 Evaluation & Discussion

In this setup, the metric of success has to do with whether or not the manipulation tasks were reproduced successfully. In addition, since its an LfD problem, to what degree of variations can observations during execution be before performance begins to degrade. There are are number of hyperparameters involved in this setup: the number of demonstrations, what interval the observation space is quantized into, the approximate matching threshold, the stage frequency used during demonstration, and whether it is constant or variable (effectively abstracting the time component out of the demonstration).

We are concerned with the robot arm being able to reproduce the demonstrated task during execution. This is assesed by observing the actions and trajectories taken, and whether or not the manipulation task demonstrated is performed successfully or not. In addition, we investigate the impact of using the approximate matching strategy on being able to acheive this as well as characterize its effects on the learned action policy.

7.1 Characterizing the learned action policies

As more demonstrations are provided to PSL, it updates the confidences of existing hypotheses or adds new hypotheses as appropriate. In other

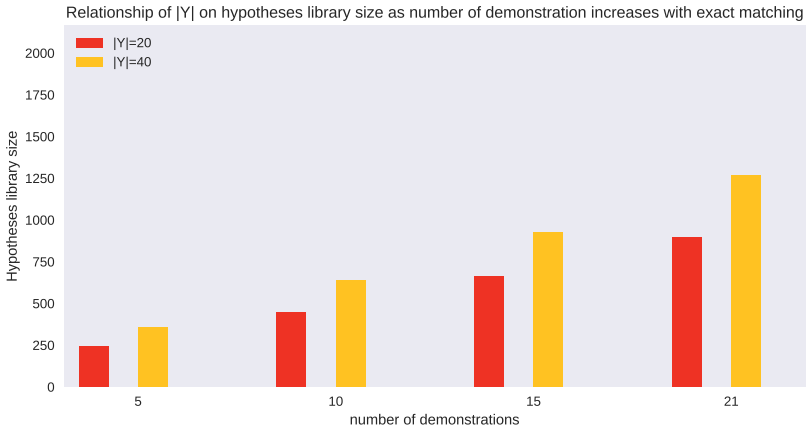

Figure 3: sub-linear growth of hypotheses library size as the number of demonstrations increases, with declining growth rate (from 82% to 46% down to 36% averaged over both values of $|Y|$). Equally larger size with larger observations space.

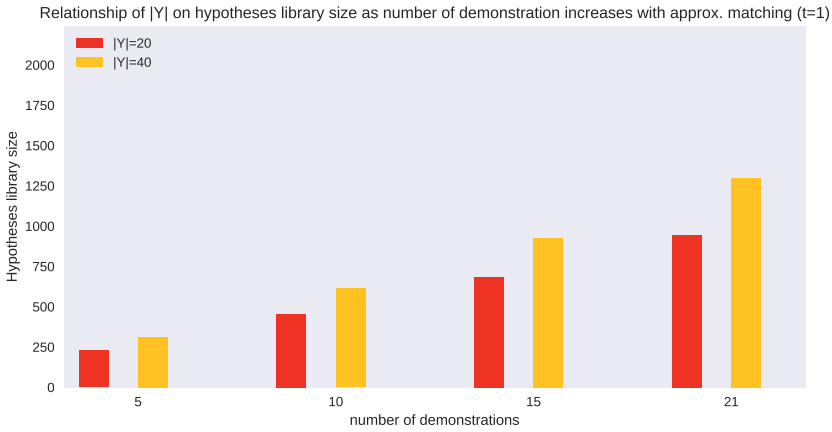

Figure 4: *Similar* sub-linear growth of hypotheses library size as the number of demonstrations increases, with declining growth rate (from 98% to 50% down to 39% averaged over both values of $|Y|$). Equally larger size with larger observations space. Approximate matching threshold set to 1.

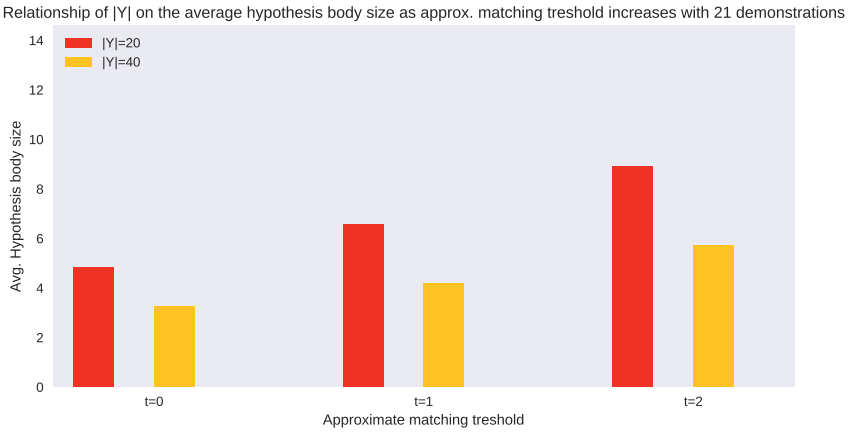

Figure 5: Approximate match has little effect on the size of the hypotheses library with the observation space sized at 20.

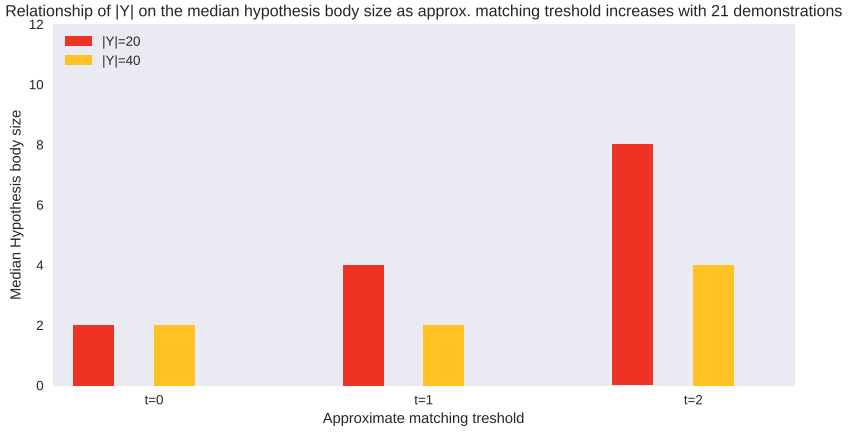

Figure 6: Approximate match exhibits the same effect on the size of the hypotheses library when the size of observation space increases.

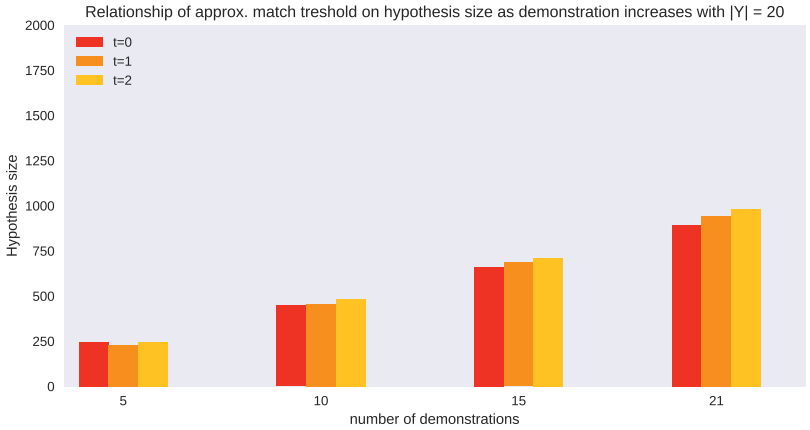

Figure 7: On Average, the individual hypothesis in the library have a body that increases as the approximate matching threshold increases. Conversely, it reduces as the observation space increases.

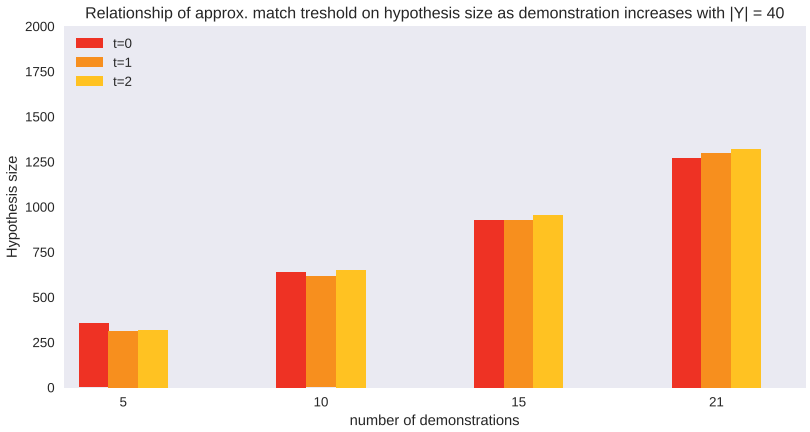

Figure 8: Similary, estimated using the median, on average, the individual hypothesis in the library have a body that increases as the approximate matching threshold increases. Conversely, it reduces as the observation space increases.

words, the learned action policies are updated and the hypotheses library is expected to keep growing until no new policy is needed based on subsequent demonstrations. The growth of the hypotheses library is shown in Figure 3. It also shows that the library increases as the size of the observation space increases. However, the growth rate of the size of the library declines with more demonstrations, indicative of the fact that less novel action policies are being learnt as the task is being repeated. The use of approximate matching does not have any effect on this property as shown in Figure 4.

In addition, the approximate matching strategy is found to have a minimal effect on the size of the hypotheses library as shown in Figure 5 and 6 which shows the impact on the hypotheses size as the matching threshold varies. However, this does not imply that no new action policies are learnt when the approximate matching strategy is used or that the learnt action policies are exactly the same as that gotten by using exact matching. In-fact, as shown in 7 and 8, the size of the body of each individual hypothesis increases as the approximate matching threshold is increased. Given the increased matching window, there is more room for ambiguity in hypothesis matching and PSL generates longer hypotheses to disambiguate the hypotheses.

After 21 demonstrations, using exact matching as well as approximate matching with thresholds of both 1 and 2, the action policy (hypotheses library) learned by PSL is able to reproduce the trajectory of the manipulation tasks successfully. However, PSL was unable to learn the right approach to the object at the right distance to grip it successfully, failing gradually as the temporal window increases. One of the reasons as to why this is so could be due to the large observation space, and the potential errors in observation from the two vision systems (the downward and sideward facing camera), making it harder to adapt correctly to variations, previously unencountered, in observation during execution. Also, since PSL continually tries to predict the next event unless a threshold is set, during execution it does so until an observation or event sequence with no match in the hypotheses library is encontered after which it gets stuck and ends.

A potential way to address the issue of sensing errors is to reduce the sensing needed to one, i.e with only one camera. As opposed to determining the pose of the end-effector through vision, we can use the camera model and inverse perspective transformation to calculate where in image space the end-effector is from its last known location in Cartesian space.

We address the need for PSL to adapt to previously unencountered states using approximate hypothesis matching. However, the based on the result, corresponding motor response (action command) may require adaptation as well. As such, the need to explicitly define a goal state arises not only as a way to prevent PSL from further prediction after the task as been executed, but also to guide the selection or vary the action command of an hypothesis. To do this, PSL would be used to predict the sensory outcome of taking an

action and the viability of taking that action is evaluated in light of the explicitly defined goal state.

Another idea is to make the robot gather data by itself using self supervision. This would ensure that a much larger state space in the sensory-motor space is encountered. The robot gathers data autonomously by determining the location of the object, again using the inverse perspective transformation, and then performing several random manipulation tasks as the sequence of observation-action pair events during this process are recorded. This data is then used to train an inverse model that predicts an intermediate action given an initial observation and a target observation. Thus, to perform a specific task, teacher demonstration would be provided as usual, but the sequence of actions during execution is predicted by the inverse model. This idea lends itself to what was done in [15] where the inverse model is regarded as providing "how to imitate" and the teacher demonstration providing "what to imitate". Considering that PSL has been used as inverse model [8] this idea makes for a potential area to be investigated.

8 Conclusion

This research project has focused on developing an action policy for a 5 degree-of-freedom robot manipulator using Predictive Sequence Learning. PSL has previously been applied exclusively to mobile robot navigation and not manipulation tasks. This report documents the theoretical model and experimental setup to adapt PSL to a pick and place task. Evaluation shows that PSL is able to learn the trajectories for the task but fails as the temporal window increases and is unable to complete the specific manipulation task of approaching the object at the right pose to grip it. We posit that this is due to the large observation space in a manipulation task and errors introduced from using two vision systems but also the need to adapt the motor command when using approximate hypothesis matching. To address this, we propose using an inverse perspective transformation to limit the sensing needed and aid a self-supervised approach to gathering demonstrations. In addition, we can use PSL to predict the sensory outcome of taking an action in form of internal simulation, effectively evaluating the hypotheses as well as adapting the action appropriately during execution. These and other strategies to achieving successful reproduction of the demonstrated tasks would be explored in the future.

References

[1] Baris Akgun, Kaushik Subramanian, and Andrea Lockerd Thomaz. Novel interaction strategies for learning from teleoperation. In *AAAI*

Fall Symposium: Robots Learning Interactively from Human Teachers, volume 12, page 07, 2012.

[2] Brenna Argall, Brett Browning, and Manuela Veloso. Learning mobile robot motion control from demonstrated primitives and human feedback. In *Robotics Research*, pages 417–432. Springer, 2011.

[3] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[4] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Handbook of Robotics*, 2008.

[5] Aude Billard, Yann Epars, Gordon Cheng, and Stefan Schaal. Discovering imitation strategies through categorization of multi-dimensional data. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2398–2403. IEEE, 2003.

[6] E. A. Billing, T. Hellström, and L.-E. Janlert. Predictive learning from demonstration. In J. Filipe, A. Fred, and B. Sharp, editors, *Proc. Second International Conference on Agents and Artificial Intelligence ICAART 2010*, volume CCIS 129, pages 186–200, Berlin Heidelberg, 2011. Springer-Verlag.

[7] E. A. Billing, T. Hellström, and L.-E. Janlert. Robot learning from demonstration using predictive sequence learning. In A. Dutta, editor, *Robotic Systems – Applications, Control and Programming*, pages 235–250. Intech, 2012.

[8] E. A. Billing, H. Svensson, R. Lowe, and T. Ziemke. Finding your way from the bed to the kitchen: reenacting and recombining sensorimotor episodes learned from human demonstration. *Frontiers in Robotics and AI*, 3(9), 2016.

[9] Erik Billing, Thomas Hellström, and Lars-Erik Janlert. Simultaneous recognition and reproduction of demonstrated behavior. *Biologically Inspired Cognitive Architectures*, 12:43–53, 2015.

[10] Erik A Billing and Thomas Hellström. A formalism for learning from demonstration. *Paladyn, Journal of Behavioral Robotics*, 1(1):1–13, 2010.

[11] Erik A Billing, Thomas Hellström, and Lars-Erik Janlert. Behavior recognition for learning from demonstration. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 866–872. IEEE, 2010.

[12] Jason Chen and Alex Zelinsky. Programing by demonstration: Coping with suboptimal teaching actions. *The International Journal of Robotics Research*, 22(5):299–319, 2003.

[13] Rüdiger Dillmann, Tamim Asfour, Martin Do, Rainer Jäkel, Alexander Kasper, Pedram Azad, Aleš Ude, Sven R Schmidt-Rohr, and Martin Lösch. Advances in robot programming by demonstration. *KI-Künstliche Intelligenz*, 24(4):295–303, 2010.

[14] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill treesrobot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

[15] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2146–2153. IEEE, 2017.

[16] Richard P Paul. Robot manipulators. *Mathematics, Programming, and Control*, 1981.

[17] Heiko Posenauer, Wolfgang Ertel, and Martin Zeller. Robot learning from demonstration by averaging trajectories, 2012.

[18] Leonel Rozo, Pablo Jiménez, and Carme Torras. A robot learning from demonstration framework to perform force-based manipulation tasks. *Intelligent service robotics*, 6(1):33–51, 2013.

[19] Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *arXiv preprint arXiv:1704.06888*, 2017.

[20] David Vernon. Machine vision-automated visual inspection and robot vision. *NASA STI/Recon Technical Report A*, 92, 1991.

[21] David Vernon. Lecture notes on cognitive robotics, 2017. Lectures 11 - 14.