

Visual Model Predictive Control

Varun Tolani



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-69

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-69.html>

May 17, 2018

Copyright © 2018, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank my principal collaborators Saurabh Gupta and Somil Bansal, without whom this work would not have happened.

Visual Model Predictive Control

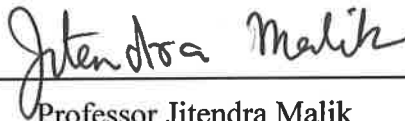
by Varun Tolani

Research Project

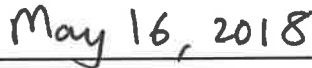
Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

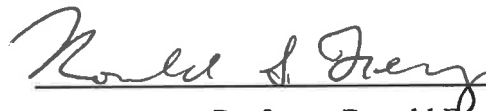
Committee:



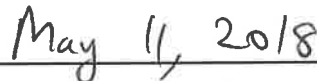
Professor Jitendra Malik
Research Advisor



(Date)



Professor Ronald Fearing
Second Reader



(Date)

Abstract

Visual Model Predictive Control

by

Varun Tolani

Masters of Science in EECS

University of California, Berkeley

Professor Jitendra Malik, Chair

We introduce an autonomous navigation framework for ground-based, mobile robots that incorporates a known dynamics model into training, allows for planning in unknown, partially observable environments, and solves the full navigation problem of goal-directed, collision-avoidant movement on a robot with complex, non-linear dynamics. We leverage visual semantics through a trained policy that, given a desired goal location and first person image of the environment, predicts a low frequency guiding control, or waypoint. We use the waypoint produced by our policy along with robust feedback controllers and known dynamics models to generate high frequency control outputs. Our approach allows for visual semantics to be learned during training while providing a simple methodology for incorporating robust dynamics models into training. Our experiments demonstrate that our method is able to reason through statistics of the visual world allowing for effective planning in unknown spaces. Additionally, we demonstrate that our formulation is robust to the particulars of low-level control, achieving performance over twice that of a comparable end-to-end learning method.

1 Introduction

This work studies the problem of indoor visual navigation for ground based mobile robots.

Classical methods for robotic navigation use a pipelined approach separating perception and control into distinct modules. The perception module localizes the robot with respect to a known map, and the control module uses this estimated location along with desired goal location to compute control commands (motor torques or velocities) to apply to the robot to move it to the desired goal location efficiently. This clean separation between perception and control led to the development of sophisticated techniques for both mapping and localization as well as planning and control. There are sophisticated techniques for going from observations (images or LiDAR scans) to 3D geometric maps [4, 29, 12], as well as advanced planning and control methods for dynamically aware navigation in these 3D maps [29]. Because these control techniques use a dynamics model to represent the underlying system, they can be designed to be robust to sensor and actuator noise as well as perturbations in the physical properties of the system. However, the choice of a purely geometric description of the world poses challenges when operating in novel environments where pre-mapping may be too expensive.

The shortcomings of the traditional methods motivates recent work in end-to-end learning-based approaches to navigation. These approaches learn policies that map raw pixels to motor controls (torques or velocities) [13, 15, 20]. Through the training process, these policies can learn about semantics, or patterns of the visual world that can enable them to operate under partial observations of the environment. However, it is challenging to leverage the known dynamics of the underlying control system with end-to-end approaches. Ignoring these dynamics makes the learning problem much harder, leading to requirement of a large number of samples to train. Moreover, these methods overfit to the underlying robot and environment dynamics and are usually not robust to even slight changes in the underlying physical system (e.g. different actuator noise models).

These drawbacks motivate recent learning-based work focused on building more generalizable navigation systems. Sadeghi et al. demonstrate transfer of simulation trained quadcopter navigation policies to real world settings [24]. Similarly recent work from Kahn et al. provides a framework for training a ground-based navigational robot in the real-world [10]. While these approaches build on generalizable learning-based navigation, they substantially simplify the navigation problem by ignoring goal-directed movement and focusing only on collision avoidance. On the other hand, Gupta et al. develop a joint mapping and planning architecture and are able to demonstrate goal-driven, collision-avoidant behavior in novel environments. However, their method also simplifies the navigation problem by assuming a discrete grid world and perfect robot egomotion [9].

Inspired by the strengths and weaknesses of the above-mentioned approaches we develop



Figure 1: Top view of our method: The robot moves from start (blue dot) to goal (green circle), periodically producing new waypoints (cyan) that guide it towards the goal while avoiding collision with obstacles (dark gray).

an intelligent controller that incorporates a known dynamics model into training, allows for planning in unknown, partially observable environments, and solves the full navigation problem of goal-directed, collision-avoidant movement on a robot with complex, non-linear dynamics. Specifically, we propose to train a policy that, given inputs of goal state and first-person-view image of the environment, outputs a waypoint that leads to collision-avoidant, goal-driven behavior. Given such a waypoint we can then use optimal control, specifically Iterative Linear Quadratic Regulator (ILQR), to interpolate smooth trajectories between the initial state, waypoint, and goal state (see Figure 1). In our method we use model predictive control, iteratively producing a waypoint, interpolating a dynamically-feasible trajectory, and moving along this trajectory for a few steps. Our method acquires semantic knowledge through the training process, applying it to plan in unknown, partially observable spaces via a low frequency guiding signal or waypoint. By incorporating known dynamics models and feedback control we hypothesize that our method can execute complex trajectories and remain robust to system changes. In section 6 we benchmark our method against a variety of baselines including a comparable end-to-end method demonstrating that our method is robust to the particulars of low-level control.

2 Related Work

Newer learning-based navigation work aims to build more generalizable, robust navigation models.

Approaches focused on real world generalizability tend to ignore the goal-driven nature of the navigation problem. Kahn et al. provide sample efficient methods for training ground base navigational robots entirely in the real world [11, 10]. Similarly, Ross et al. provides a solution for training sensor limited quadcopters to fly [23]. Sadeghi et al. demonstrate transfer of simulation based policies for quadcopter navigation to a real-world quadcopter [24]. Finally, work from Gandhi et al. proposes yet another novel solution to training policies for quadcopter flight entirely in the real world [6]. While these methods build on generalizability of learning-based navigation techniques, they entirely ignore the goal driven aspect of the navigation problem.

On the other hand, Richter et al. achieve goal-driven, collision-avoidant navigation while incorporating robot dynamics models and generalizing across systems and unknown environments (i.e. trained in simulation and deployed in the real world) [22, 21]. Their work, however, assumes access to a laser scanner for constructing high quality 3d belief maps on the fly, and learns a logistic regression model on top of hand-designed features (e.g. estimated distance to nearest obstacle, estimated velocity to nearest obstacle, etc.). In contrast, we do not construct an intermediate map representation, but learn directly from RGB imagery. We use a convolutional neural network which eliminates the need for hand designed features. Additionally, while planning over a short horizon, Richter et al. explicitly consider a variety of dynamically feasible trajectories, selecting and executing the lowest cost one (their cost function penalizes collision and encourages goal directed movement). In contrast, we assume collision-free behavior over a sufficiently small horizon and use ILQR to plan over this horizon.

Simulation based approaches, on the other hand, tend to focus only on goal-driven behavior, simplify complex robot dynamics, or even ignore them entirely. Work in learned mapping representations from Parisotto et al. assumes a discrete action space and ignores collision avoidance [19]. Recent work from Mirowski et al. demonstrates autonomous robot navigation through cities, but assumes an underlying discrete graph structure, simple robot dynamics, perfect robot egomotion, and entirely ignores the collision-avoidance problem [18]. In contrast, Gupta et al. focus on goal-driven and collision-avoidant navigation, but similarly assume simplified robot dynamics, a grid world, and perfect egomotion [9].

Our work attempts to bridge these bodies of work, providing an intelligent low level controller that observes the environment through first-person RGB images and produces goal-driven, collision-avoidant behavior in a dynamically aware fashion.

3 Background

In this section we elaborate on core algorithms and techniques used in our framework.

Iterative Linear Quadratic Regulator (ILQR)

Optimal control aims to find the optimal trajectory, τ^* , with respect to a cost function c_{ILQR} subject to the constraint that τ^* is dynamically feasible with respect to a dynamics model f . Let \mathbb{J} denote the following optimal control problem:

$$\mathbb{J} = \min_{[(s_0, u_0) \dots (s_T, u_T), s_{T+1}]} c_{ILQR}(s_{T+1}) + \sum_{t=0}^T c_{ILQR}(s_t, u_t) : s_{t+1} = f(s_t, u_t) \quad \forall t \in [0, T]$$

$$\tau^* = \arg \min_{[(s_0, u_0) \dots (s_T, u_T), s_{T+1}]} c_{ILQR}(s_{T+1}) + \sum_{t=0}^T c_{ILQR}(s_t, u_t) : s_{t+1} = f(s_t, u_t) \quad \forall t \in [0, T]$$

If c_{ILQR} is quadratic and f is linear the Linear Quadratic Regulator(LQR) provides a dynamic programming solution to exactly solve the optimal control problem. For problems where f is non linear and/or c_{ILQR} non quadratic, ILQR provides an iterative approximation method leveraging LQR to solve local approximations to \mathbb{J} . Let $\tau_i = [(s_0^i, u_0^i) \dots (s_{T_i}^i, u_{T_i}^i), (s_{T_{i+1}}^i)]$ denote a trajectory around which we initially linearize to run ILQR. For a horizon of T , ILQR solves for $K_t, k_t \quad \forall t \in [0, T]$ which dictate locally optimal linear feedback controllers. The optimal control at each time step $t \in [0, T]$ can then be calculated in the following manner [28]:

$$u_t^* = K_t(s - s_t^i) + \alpha k_t + u_t^i$$

Here $K_t(s - s_t^i)$ is the feedback term, k_t is the feedforward term, u_t^i is the reference term, and α is a step size over which we line search (starting from $\alpha = 1$). In our formulation the inputs to ILQR are f , a dynamics model, c_{ILQR} , a cost function, τ_i , an initial trajectory, τ_R , a reference trajectory against which to penalize, and n , the number of iterations for which we run ILQR. Pseudocode for ILQR is presented in the appendix.

Fast Marching Method

The Fast Marching Method (FMM) aims to solve the Eikonal Equation, a partial differential equation which describes the evolution of the surface $t(s)$ with speed $f(s)$ in the normal direction to the surface $t(s)$ [25].

$$|\nabla t(s)| = \frac{1}{f(s)}$$

$$\text{such that: } s \in \Omega, t(\delta\Omega) = 0, f(s) > 0 \quad \forall s \in \Omega$$

When $s \in \mathbb{R}^2$, the function $t(s)$ can be thought of as the time to reach $\delta\Omega$ from s moving at speed $f(s)$. Let \mathbb{S} be the Cartesian plane discretized in units of size δx and \tilde{s} be

the discretized version of s that lies on \mathbb{S} . FMM iteratively uses a numerical differencing approach to approximately solve the Eikonal Equation in a manner that closely resembles Dijkstra’s algorithm for shortest paths. We use $T_{FMM} = FMM_TIME(\delta\Omega)$ to denote the output of the Fast Marching Method, a discretized grid where $T_{FMM}(\tilde{s})$ represents the time to reach the goal $\delta\Omega$ from \tilde{s} moving at speed $f(\tilde{s})$. In our method the speed function is constant for all \tilde{s} . In particular we use $v = f(\tilde{s}) \forall \tilde{s} \in \mathbb{S}$. Thus a simple relation between FMM_DIST and FMM_TIME holds, specifically:

$$FMM_DIST(\tilde{s}) = FMM_TIME(\tilde{s}) * v = T_{FMM}[\tilde{s}] * v$$

We use FMM to find shortest feasible paths to the goal in our environment. To interpolate the shortest path from a state s_t to the goal $\delta\Omega$ we follow the direction of the negative gradient of $t(s)$ evaluated at s_t (direction of steepest descent). As FMM is a discrete method this amounts to iteratively applying the update:

$$s_{t+1} = s_t - \Delta(\nabla T_{FMM}(\tilde{s}_t)) \text{ for a small enough } \Delta$$

Pseudocode for FMM is presented in the appendix.

Convolutional Neural Networks

In the computer vision community convolution has been a widely used tool for image understanding, feature extraction, filtering, etc. as it is known to exploit local spatial correlation of pixels while providing some amount of spatial invariance. Common image processing pipelines have alternated convolution and subsampling, allowing for reasoning over images at various spatial resolutions, since the later 20’t century [1, 3]. Convolutional Neural Networks (CNN), build on this work, taking a data-driven approach, allowing optimal filters to be learned from data. Typical CNN’s alternate convolutional layers with non-linear activation functions and subsampling layers.

Some of the first major work in data-driven, learned convolutional filters was released in 1980 with the advent of the Neocognitron by Fukushima et al. [5]. In 1998 Yann LeCun expanded on this work by training a CNN using backpropogation on the task of hand-written digit classification [17]. Though these advances in neural networks came at the end of the 20’t century, CNN’s did not gain popularity until recent advances in “big data” and hardware made them feasible. CNN’s now provide state of the art results in many computer vision tasks [7, 16].

4 Our Approach

The robot is placed into a new, unknown environment and given a goal location (g) specified in its egocentric coordinate frame. At each time step the robot observes the environment through first person images (I_t). Its goal is to navigate efficiently through the unknown environment towards the goal while avoiding collision. We define the robot’s trajectory τ , of length T , as a collection of dynamically feasible states (s_t) and actions (u_t), with respect to the robot’s dynamics model (f).

$$\begin{aligned} \tau &= [(s_0, u_0), \dots, (s_T, u_T), (s_{T+1})] \\ &\text{subject to: } s_{t+1} = f(s^t, u^t) \end{aligned}$$

Parameterized Model-based Controller

Rather than move directly towards the goal, our robot moves towards an intermediate point, or waypoint (w_t). Given w_t and g we use a heuristic trajectory mapper ($H(w_t, g)$) to construct an infeasible reference trajectory ($\tau_{w_t}^R$) between the robot’s current location, waypoint, and goal. We use ILQR with a cost function (c_{ILQR}) to generate a low-cost, dynamically feasible trajectory, τ_{w_t} , around $\tau_{w_t}^R$. The robot then takes h steps along τ_{w_t} before observing a new image I_{t+h} , producing a new waypoint w_{t+h} , and repeating the process. The robot’s trajectory, τ , in terms of these intermediate waypoint driven trajectories is:

$$\tau = [\tau_{w_0}[0 : h - 1], \tau_{w_h}[h - 1 : 2h - 1], \dots]$$

Learning To Predict Waypoints

We pose the problem of choosing a goal-directed, collision-avoidant waypoint as a classification problem over N_w waypoints in the set \mathbb{W} . We train a parametrized policy $\pi_\theta(I_t, g)$ to predict optimal waypoints. To train π_θ we first compute ground truth free space maps in our simulated environment. We then compute shortest collision-free paths through the map using FMM. For each waypoint $w_t \in \mathbb{W}$ we compute a heuristic reference trajectory $\tau_{w_t}^R = H(w_t, g)$, then computing \mathbb{W}_{CF} , the set of waypoints that correspond to collision free reference trajectories:

$$\mathbb{W}_{CF} = \{w_t | \text{No collision along } \tau_{w_t}^R\}$$

For each waypoint w_t in \mathbb{W}_C we then compute the FMM_COST over the first h steps of τ_{w_t} as the weighted sum of the FMM distance along the trajectory and the alignment to the FMM gradient (shortest feasible path to the goal).

$$\text{FMM_COST}(w_t) = \frac{1}{h}(\text{FMM_DIST}(\tau_{w_t}[:h]) + \lambda \text{FMM_ALIGNMENT}(\tau_{w_t}[:h]))$$

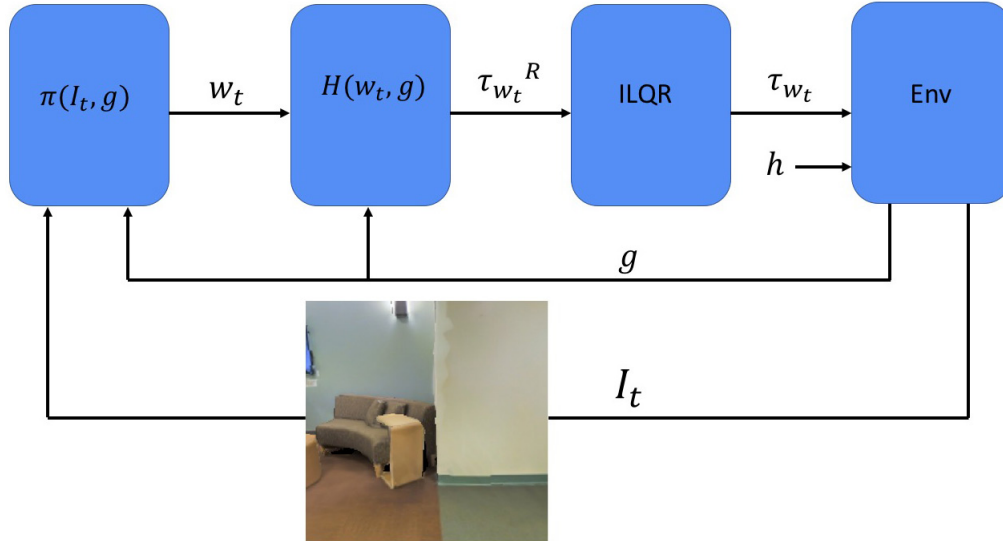


Figure 2: From left to right: 1. The robot uses policy $\pi_\theta(I_t, g)$ to produce a waypoint w_t . 2. The robot uses a heuristic trajectory mapper $H(w_t, g)$ to plan an infeasible reference trajectory, $\tau_{w_t}^R$, between start, waypoint, and goal. 3. The robot uses ILQR along with cost function c_{ILQR} to generate a feasible reference trajectory τ_{w_t} . 4. The agent executes τ_{w_t} in the environment for h steps then makes a new observation I_{t+h}

The optimal waypoint selected for supervision is then:

$$w_t^* = \arg \min_{w_t \in \mathbb{W}} \text{FMM_COST}(w_t)$$

Pseudocode for our algorithm is given below:

```

1 Our Algorithm:
2   #Collect Data
3   inputs, outputs = [], []
4   while i < N_D:
5     env.reset() #samples a new start and goal
6     done = False
7     while not done:
8       inputs.append((env.get_obs(), env.goal()))
9       opt_waypt = calculate_optimal_fmm_waypt(env.goal(), lambda, h)
10      outputs.append(opt_waypt)
11      next_state, done = env.step(opt_waypt) #use ILQR and take h steps
           along this path
12      i++
13
14   #Train pi
15   D = (inputs, outputs)
16   D = standardize(D)

```

```

17     train, valid = split(D)
18      $\pi$  = setup and initialize network
19     for i in [0, num_epochs]:
20         for batch in batches:
21             train  $\pi$  on batch
22     return  $\pi$ 

1 calculate_optimal_fmm_waypt(g,  $\lambda$ , h):
2     waypt_trajs = []
3     for  $w_i \in \mathbb{W}$ :
4          $\tau_{w_i} = H(w_i, g)$  #heuristic trajectory through  $w_i$  and  $g$ 
5         if (No collision along  $\tau_{w_i}$ ):
6             waypt_trajs.append( $\tau_{w_i}[0:h]$ ) #follow this for  $h$  steps
7     dists, alignments = [], []
8     for traj in waypt_trajs:
9         dist_cost, align_cost = 0, 0
10        for  $s_t \in \text{traj}$ :
11            fmm_grad_x, fmm_grad_y =  $\nabla_x$  FMM.TIME,  $\nabla_y$  FMM.TIME
12            fmm_heading = arctan2(fmm_grad_y[ $s_t$ ], fmm_grad_x[ $s_t$ ])
13            robot_heading = arctan2( $s_t[1]$ ,  $s_t[0]$ )
14            align_cost += wrap(robot_heading, fmm_heading)
15            dist_cost += FMM.DIST[ $s_t$ ]
16        dists.append(avg(dist_cost))
17        alignments.append(avg(align_cost))
18    return argmin(dists +  $\lambda$ *alignments)

```

5 Experimental Setup

Agent Setup

We model our robot in simulation as a discrete time “augmented” Dubins Car. The augmented Dubins Car formulation has a state space of $s = [x, y, \theta, \tilde{v}, \tilde{\omega}]$ and control space of $u = [\Delta v, \Delta \omega]$. Here \tilde{v} and $\tilde{\omega}$ represent unsaturated linear and angular velocity and sat_1, sat_2 represent saturation functions for linear and angular velocity respectively. The augmented Dubins Car is functionally equivalent to the ideal Dubins Car (see appendix), but allows our ILQR cost function to penalize linear and angular acceleration, rather than velocity of the robot leading to smoother trajectories.

Augmented Dubins Car Dynamics Model

$$\begin{bmatrix} x \\ y \\ \theta \\ \tilde{v} \\ \tilde{\omega} \end{bmatrix}_{t+1} = f(\vec{s}_t, \vec{u}_t) = \begin{bmatrix} x_t + \Delta t * \cos(\theta_t) * sat_1(\tilde{v}_t) \\ y_t + \Delta t * \sin(\theta_t) * sat_1(\tilde{v}_t) \\ \theta_t + \Delta t * sat_2(\tilde{\omega}_t) \\ \Delta \tilde{v}_t + \tilde{v}_t \\ \Delta \tilde{\omega}_t + \tilde{\omega}_t \end{bmatrix}$$

We model our agent as a cylinder of height .8m and radius .15m. The robot has a RGB camera mounted at height .8m with 120 degree horizontal and vertical field of view. The camera is tilted 15 degrees below the horizontal. We use linear clipping for saturation functions sat_1, sat_2 clipping the linear velocity to be in the range $[0, .55]$ m/s and the angular velocity to be in the ranges $[-1.1, 1.1]$ rad/s. For all experiments we use $\Delta t = .1$ seconds.

ILQR Parametrization

In our ILQR implementation we use a quadratic cost function c_{ILQR} designed to minimize the weighted squared distance from a given reference trajectory τ_R .

$$\begin{aligned} c_{ILQR}(s_t, u_t) &= (s_t - s_t^R)^T Q (s_t - s_t^R) + (u_t - u_t^R)^T R (u_t - u_t^R) \\ Q &= \text{diag}([\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]) \\ R &= \text{diag}([\alpha_6, \alpha_7]) \end{aligned}$$

Here Q and R are square matrices with elements α_i on the diagonals and zeros elsewhere. We fix $\alpha_1 = \alpha_2 = \alpha_3 = 4.0$, $\alpha_4 = \alpha_5 = 1e - 5$, and systematically vary α_6, α_7 in our experiments.

Waypoints

To allow for translational as well as pure rotational behavior we represent our waypoint grid \mathbb{W} as the union of two waypoint subsets, \mathbb{W}_R , a set of purely rotational waypoints (i.e. of the form $[0, 0, \theta]$) and \mathbb{W}_T , a set of translational waypoints equally distributed around a conical field of view centered at the robots camera center. To generate \mathbb{W}_T we uniformly sample a 7x7 grid in polar coordinates with $r \in [0, 2.0]$ meters and $\theta \in [-30, 30]$ degrees (see figure 3). To construct \mathbb{W}_R we uniformly sample 11 angles in the range $\theta \in [-30, 30]$.

Simulation Environment

We train and evaluate our approach using the Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) which contains 6 different indoor building environments rendered from scans of real Stanford buildings using a Matterport camera [2]. In all experiments we set our problem horizon, T , to be 200 and our waypoint horizon h to be 20. We consider a circle of radius .3 meters around the goal to be the “success region”. Upon reaching the “success region” we immediately terminate the episode. We call a trajectory τ a “success” if the robot reaches the “success” region in under $T = 200$ steps without collision.

Let s_G and g_G denote a start and goal position in the global coordinate frame respectively. We find that randomly sampling s_G and g_G in free space resulted in many navigation problems that could either be solved by taking a straight line path to the goal or were unrealistically difficult to expect the robot to solve in 200 time steps. In our final method for

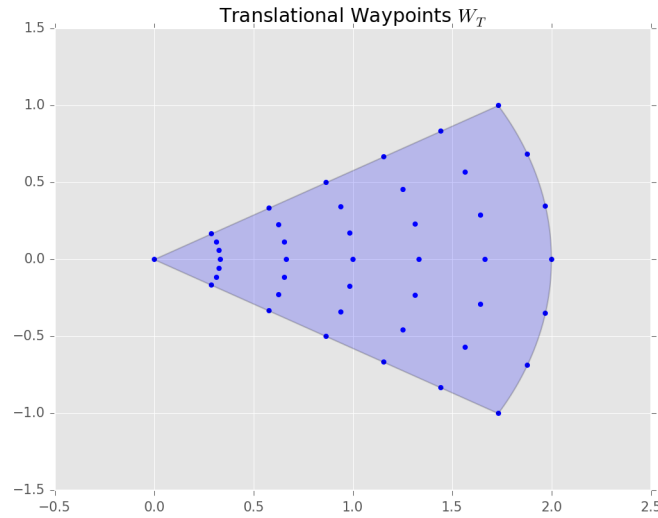


Figure 3: Equally spaced translational waypoints, $w \in \mathbb{W}_T$, for a robot at $[0,0]$ facing along the x axis. Waypoints are uniformly sampled in a conical field of view using polar coordinates with $r \in [0.0, 2.0]$ meters and $\theta \in [-30, 30]$ degrees.



Figure 4: Example of data from the S3DIS dataset. Left: First-person view rendered from the robot's perspective. Right: Corresponding topview of the environment from the robot's perspective (black arrow). Here light gray represents free space, while dark gray represents occupied space (obstacles).

sampling navigation problems, we sample s_G from our precomputed free space map. We then randomly sample a distance d_i between $[0, .5]$ meters. The goal g_G is then sampled from the set \mathbb{G} :

$$\mathbb{G} = \{g_G \mid \|FMM_DIST(g_G, s_G) - \|g_G - s_G\|_2\|_2 \geq d_i, .3 \leq \|g_G - s_G\|_2 \leq 5\}$$

We use the above FMM-L2 heuristic to choose a suitably difficult distribution of problems where the shortest feasible paths between start and goal are larger than the straight line path by at least some distance d_i . In other words our sampling procedure selects “interesting” problems where the agent cannot simply move in a straight line, but must navigate around obstacles. All training, validation, and test problems are sampled in this manner.

Network Architecture

We represent our policy function as a CNN. The output of our policy is:

$$w_t = \pi_\theta(I_t, g) = \arg \max_{w \in \mathbb{W}} \text{softmax}(\phi_3([\phi_1(I_t), \phi_2([g])]))$$

Here ϕ_1 is a learned image encoder represented by 5 convolution, rectified linear unit (ReLU), max-pooling blocks, ϕ_2 is a learned goal encoding represented by a single fully connected layer with ReLU activation, and ϕ_3 is a 3 layer multilayer perceptron (MLP) with ReLU activation functions that produces logits of dimension N_w . We use a cross entropy loss in specifying our objective function and train our network using the ADAM optimizer with a batch size of 64 and a learning rate of $5e - 4$ [14]. We initialize our network with Xavier initializer, which is designed keep the variance of the input and output of each network layer constant [8].

Deep neural networks often have many more parameters than available data, leading to overfitting [27]. We use standard techniques to avoid overfitting in deep neural networks including dropout (with dropout probability .15 in the second to last MLP layer) [27], weight decay (using l2 norm and regularizer strength $1e - 5$) [16], and data augmentation (randomly adjusting brightness and saturation of RGB images during training) [26]. We use a 80%, 20% training, validation split using cross validation to select all hyper parameters.

We keep a held out set of 200 goals in each of the 6 S3DIS environments on which we test our agent. We train our agent on approximately 6000 episodes of data from 1 S3DIS environment, keeping 1 S3DIS environment for validation and testing respectively. All neural networks for our method and baselines are trained for a maximum of 28 epochs. We report all metrics recorded when validation loss is lowest (typically around 10-12 epochs).

Baselines

We compare our method against a variety of methods:

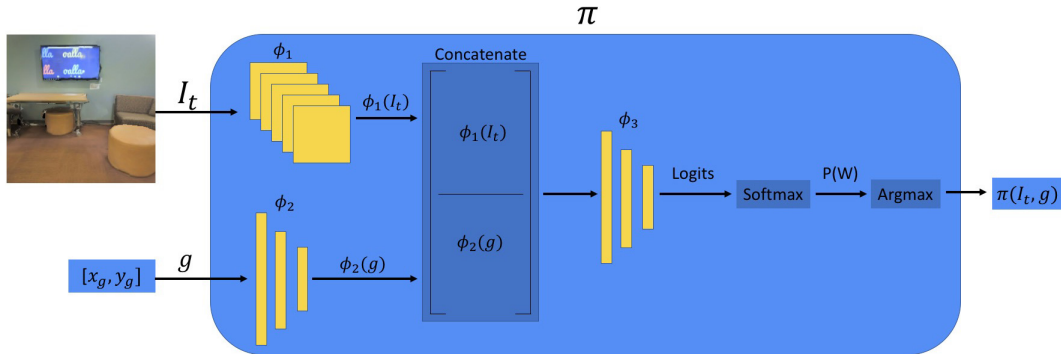


Figure 5: Policy Architecture: The policy π processes a first person image, I_t and the robot’s current goal (specified in egocentric coordinates), g , predicting a probability distribution over the \mathbb{N}_w waypoints, and selecting the index of the waypoint with the maximum probability.

Random Waypoint Baseline (Random Wpt)

The policy $\pi_\theta(I_t, g)$ in this method samples a waypoint w_t at random from \mathbb{W} , executing it for h steps.

No Waypoint Baseline (No Wpt)

We compare against a waypoint-less baseline, which simply turns towards the goal and proceeds in a straight line until reaching the “success” region or colliding with an obstacle.

No Image Baseline (No Image)

We compare our method against a similar, but visionless method. We remove the learned image encoder and train our visionless policy $\pi_\theta(g)$ using the same settings as in “our” method. We represent our visionless policy as:

$$w_t = \pi_\theta(g) = \arg \max_{w \in \mathbb{W}} \text{softmax}(\phi_3(\phi_2([g])))$$

End-to-end, Discrete Action Space, FMM Supervision (FMM Disc)

We compare our method against a comparable end-to-end approach, which we term FMM Disc. FMM Disc learns a policy $\pi_\theta(I_t, g)$ that outputs a raw control command at each time step (u_t).

$$u_t = [\Delta \tilde{v}_t, \Delta \tilde{w}_t]$$

We frame the action selection problem posed by FMM Disc as a classification problem over \mathbb{N}_u actions where $\mathbb{N}_u = 49$. To generate supervision for FMM Disc we first use `calculate_optimal_fmm_waypt` (section 44) to compute a waypoint w_t at each timestep t , then

computing a heuristic trajectory $\tau_{w_t}^R$ with our heuristic trajectory planner $H(w_t, g)$. Next we apply ILQR to reference trajectory $\tau_{w_t}^R$ creating a dynamically feasible reference trajectory τ_{w_t} . Finally we discretize the first action u of τ_{w_t} into one of N_u bins. We use this discretized action as our supervision signal.

6 Results

We report 3 metrics- mean final distance to goal, collision rate, and success rate over a held out set of 200 navigational goals in training, validation, and testing environments. To demonstrate our method’s robustness to the particulars of low level control, we run our tests with two different ILQR cost functions c_{ILQR}^L , a low control cost version with $\alpha_6 = \alpha_7 = 1e-5$ and c_{ILQR}^H , a high control cost version with $\alpha_6 = \alpha_7 = 1$. Results are presented in tables 1 and 2. Both our method and FMM DISC perform noticeably better, with respect to all three metrics, in the training environment than in the validation or test environments; thus we focus our analysis on metrics in the test environment.

Table 1: Low Control Penalty

	Train			Validation			Test		
	Coll %	Final Dist	Success Rate	Coll Rate	Final Dist	Success Rate	Coll Rate	Final Dist	Success Rate
Wpt	.09	.40	.875	.17	.66	.74	.13	.86	.665
FMM Disc	.06	.391	.905	.23	.62	.725	.15	.49	.8
No Image	.87	1.96	.12	.77	1.68	.22	.77	1.8	.21
Random	.95	3.41	.01	.955	3.48	.95	.975	3.59	0.01
Wpt									
No Wpt	.965	2.055	.035	.96	1.89	.04	.99	2.01	.02

In table 1 we find that our method outperforms the No Image, Random Wpt, and No Wpt baselines while performing only slightly worse than the end-to-end method, FMM Disc. We conclude that in the low control cost setting, FMM DISC is able to learn to successfully navigate slightly better than our method in terms of final distance (.49 vs .86 meters) and success rate (.8 vs .665). Comparing methods across tables 1 and 2 we find, however, that our method remains robust to the particulars of low level control (similar final distances (.86 vs .74), collision rates (.13 vs .165), and success rates (.665 vs .715)). The FMM Disc baseline, on the other hand, scores substantially worse when trained using c_{ILQR}^H . The FMM Disc method achieves a final distance 3 times worse (1.5 vs .49) and collision rate almost three times as high (.49 vs .15) when trained using c_{ILQR}^H .

Table 2: High Control Penalty

	Train			Validation			Test		
	Coll %	Final Dist	Success Rate	Coll Rate	Final Dist	Success Rate	Coll Rate	Final Dist	Success Rate
Wpt	.06	.412	.89	.145	.578	.785	.165	.74	.715
FMM Disc	.24	.83	.70	.33	1.01	.55	.39	1.53	.55
No Image	.86	1.90	.14	.80	1.7	.2	.83	1.90	.18
Random	.97	3.48	0	.985	3.31	0	.935	3.6	0.015
Wpt									
No Wpt	.97	2.07	.035	.995	1.89	.045	.98	2.01	.02

In both tables 1 and 2 our method vastly outperforms the No Image, Random Wpt, and No Wpt baselines. In the low control penalty setting the No Wpt baseline achieves .99 collision rate, indicating that the majority of the held-out test goals are not solveable by simply following a straight line path to the goal. The same pattern holds true in the high control penalty setting with the No Wpt baseline colliding 98% of the time. As our method achieves a .13 and .15 collision rate in these same test cases, we conclude that our method dramatically outperforms a simple, greedy straight line heuristic.

In comparison to the No Image baseline in table 1 we see that our method reduces empirical collision rate by a factor of almost 6 (.13 vs .77), and average final distance by a factor of 2 (.86 meters vs 1.8 meters). We note that this pattern holds across known and unknown environments indicating that our method has learned to reason through the statistics of the visual world. In figure 6 we visualize a particular trajectory from our trained policy in which the robot makes a semantically based decision to navigate through a doorway to reach the goal. In figure 7 we visualize the variance of our method as we vary the random seed used in our environment. In figure 8 we visualize topviews for four different successful goals using our waypoint method.

7 Conclusion

Inspired by the strengths and weaknesses of traditional, and learning based navigation approaches we propose a new method designed to bridge the bodies of work. Our method uses tools from optimal control and deep learning to allow the robot to learn semantics of the visual world while allowing for simple incorporation of known dynamics models. We hypothesize and demonstrate empirically that learning a low frequency guiding control (waypoint) allows our system to remain independent of the particulars of low level control as compared to a comparable end-to-end method. In terms of final distance from goal and percent col-

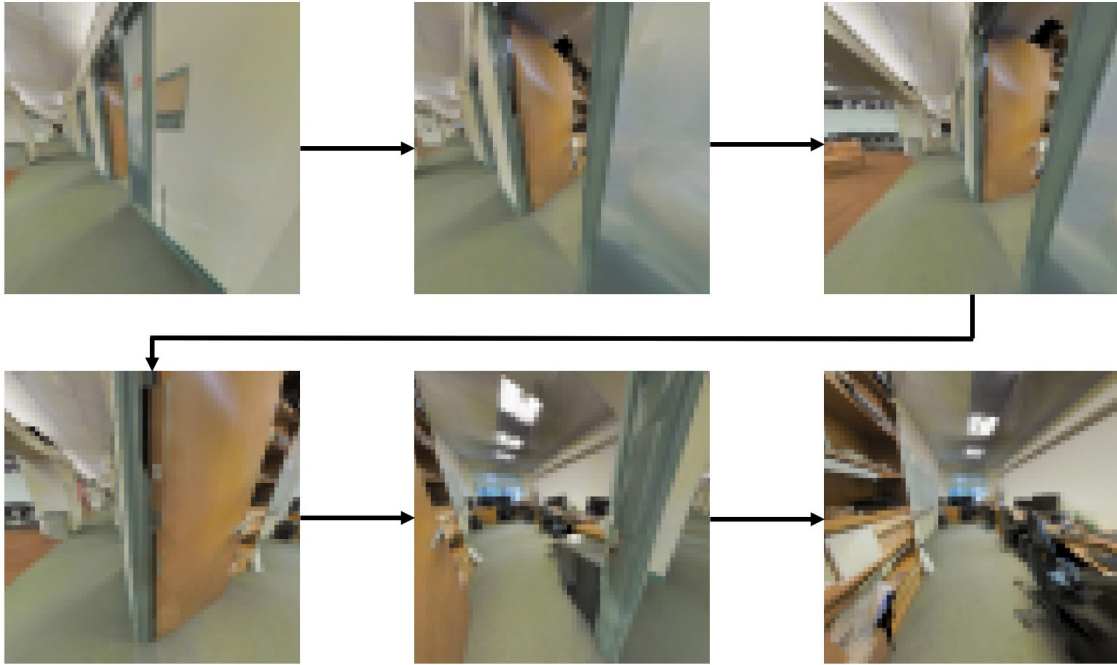


Figure 6: Visualized trajectory of our trained robot navigating from a hallway to a goal inside an office room. Here the robot must guide itself to and through an open doorway, though it is not immediately obvious that doing so will lead till the goal. We conclude that the robot has learned some semantics of the visual world through the training process.

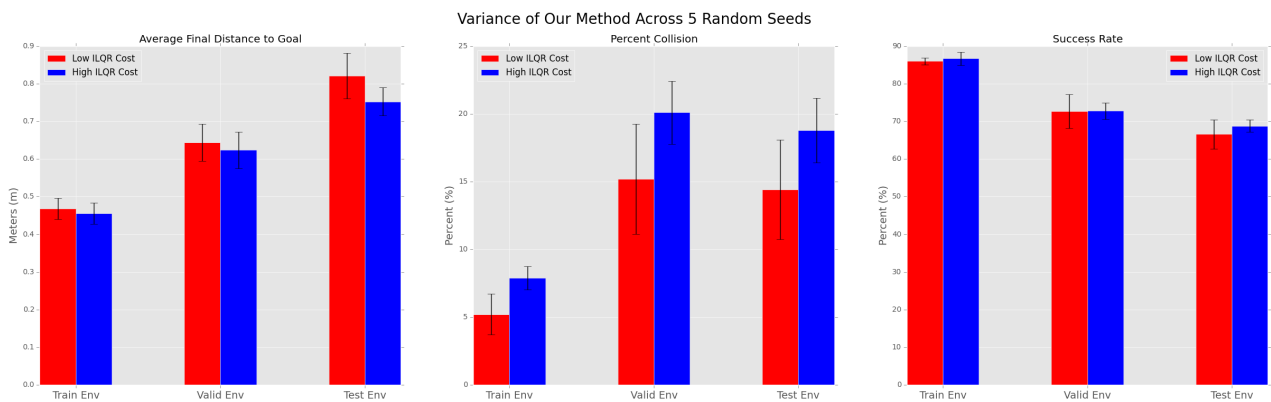


Figure 7: Variance of the metrics average collision rate, final distance to goal, and success rate of our method in training, validation, and testing environments as we vary the random seed. We plot the mean and variance of the metrics across 5 different random seeds for both the c_{ILQR}^L and c_{ILQR}^H methods.

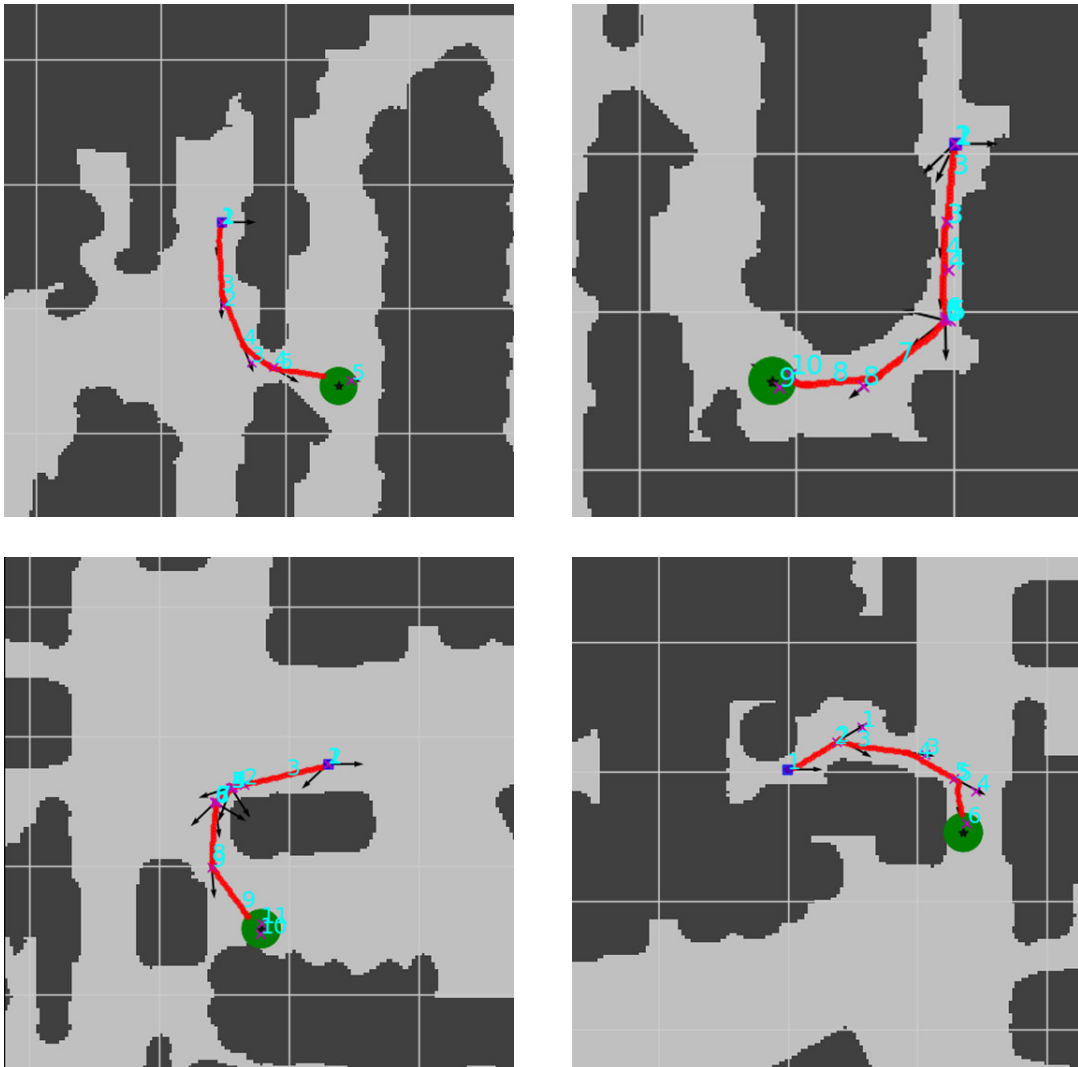


Figure 8: Topviews of 4 successful trajectories (red) generated using our waypoint method. The robot successfully navigates from start (blue dot) to goal (green circle) while avoiding collision (dark gray) using a series of guiding waypoints (cyan).

lisions, our proposed method achieves performance more than twice that of a comparable end-to-end method.

We also demonstrate that our method is able to effectively reason through statistics of the visual world as it dramatically outperforms a vision-less baseline by a factor of 6 (in terms of collision rate). This performance holds across a diverse set of indoor simulation environments both seen and unseen during training indicating that our method has not merely overfit to the data, but rather learned to reason through patterns of the visual world. Finally, we visualize a particular trajectory where our robot demonstrates semantic understanding of the visual world through navigation.

We provide our framework as a method for building a generalizable, learning-based, intelligent controller that incorporates semantic reasoning into dynamically aware path planning in unknown, partially-observable environments.

Appendix

ILQR Pseudocode:

```

1 def ILQR(f, c,  $\tau_i$ ,  $\tau_R$ , n):
2   Loop n times:
3     cost = c( $\tau_i, \tau_R$ )  #(cost of  $\tau_i$  wrt  $\tau_R$ )
4      $A_t, B_t, Q_t, R_t$  = 1st order approximation of f and 2nd order
5                       approximation of c around  $\tau_i \forall t \in [0, T]$ 
6     Calculate  $K_t, k_t = \text{LQR}(A, B, Q, R)$ 
7      $\alpha = 1$ 
8     Loop: #Line Search Over  $\alpha$ 
9        $u^{t*} = K_t(s - s_1^t) + \alpha k_t + u_1^t \forall t \in [0, T]$ 
10       $s^* = \text{apply actions } u^{t*} \text{ through system } f \text{ starting at } s_i^0$ 
11       $\tau = [s^*, u^*]$ 
12      new_cost = c( $\tau, \tau_R$ )
13      if new_cost < cost:
14         $\tau_i = \tau$ 
15        break
16      else:
17        decrease alpha
18  return  $\tau_i$ 

```

FMM Pseudocode:

```

1 def FMMTIME( $\delta\Omega$ ,  $\delta x$ ):
2   Discretize the space in intervals of  $\delta x$ 
3   For every  $\tilde{s}_i \in \mathbb{S}$ :
4      $T_{FMM}(\tilde{s}_i) = \infty$ 
5     label( $\tilde{s}_i$ ) = far
6    $T_{FMM}(\delta\Omega) = 0$ 
7   label( $\delta\Omega$ ) = accepted
8   Loop:
9     For every  $\tilde{s}_i \in \mathbb{S}$ :
10       $\tilde{T}_i = \text{eikonal\_update}(\tilde{s}_i, dx)$ 
11      if  $\tilde{T}_i < T_{FMM}(\tilde{s}_i)$ :
12         $T_{FMM}(\tilde{s}_i) = \tilde{T}_i$ 
13        label( $\tilde{s}_i$ ) = considered
14    $S_c = \{\tilde{s}_i | \text{label}(\tilde{s}_i) = \text{considered}\}$ 

```

```

15      $\tilde{s} = \arg \min_{\tilde{s}_i \in S_c} T_{FMM}(\tilde{s}_i)$ 
16     label( $\tilde{s}$ ) = accepted
17     For every neighbor  $\tilde{s}_i$  of  $\tilde{s}$ :
18         if label( $\tilde{s}_i$ ) != accepted:
19              $\tilde{U}_i = \text{eikonal\_update}(\tilde{s}_i, dx)$ 
20             if  $\tilde{U}_i < U(\tilde{s}_i)$ :
21                  $T_{FMM}(\tilde{s}_i) = \tilde{U}_i$ 
22                 label( $\tilde{s}_i$ ) = considered
23      $S_c = \{\tilde{s}_i | \text{label}(\tilde{s}_i) = \text{considered}\}$ 
24     if len( $S_c$ ) == 0:
25         break
26     return  $T_{FMM}$ 

1 #solve a discrete approximation to the Eikonal Equation
2 def eikonal_update( $\tilde{s}_i, dx$ ):
3      $x, y = \tilde{s}_i$ 
4      $T_H = \min(T_{FMM}[x-1, y], T_{FMM}[x+1, y])$ 
5      $T_V = \min(T_{FMM}[x, y-1], T_{FMM}[x, y+1])$ 
6      $T_{FMM} = \frac{T_H + T_V}{2} + \frac{1}{2} \sqrt{(T_H + T_V)^2 - 2(T_H^2 + V_H^2 - \frac{dx^2}{f(\tilde{s}_i)})}$ 
7     return  $T_{FMM}$ 

```

Ideal Dubins Car Dynamics Model

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t+1} = f(s_t, u_t) = \begin{bmatrix} x_t + \Delta t * \cos(\theta_t) * v_t \\ y_t + \Delta t * \sin(\theta_t) * v_t \\ \theta_t + \Delta t * \omega_t \end{bmatrix}$$

$$s_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix}$$

Bibliography

- [1] Edward H Adelson et al. “Pyramid methods in image processing”. In: *RCA engineer* 29.6 (1984), pp. 33–41.
- [2] Iro Armeni et al. “3D Semantic Parsing of Large-Scale Indoor Spaces”. In: *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. 2016.
- [3] Peter J Burt and Edward H Adelson. “The Laplacian pyramid as a compact image code”. In: *Readings in Computer Vision*. Elsevier, 1987, pp. 671–679.
- [4] Andrew J Davison and David W Murray. “Mobile robot localisation using active vision”. In: *European Conference on Computer Vision*. Springer. 1998, pp. 809–825.
- [5] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.
- [6] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. “Learning to Fly by Crashing”. In: *CoRR* abs/1704.05588 (2017). arXiv: 1704.05588. URL: <http://arxiv.org/abs/1704.05588>.
- [7] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.
- [8] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feed-forward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.
- [9] Saurabh Gupta et al. “Cognitive mapping and planning for visual navigation”. In: *arXiv preprint arXiv:1702.03920* 3 (2017).
- [10] Gregory Kahn et al. “Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation”. In: *arXiv preprint arXiv:1709.10489* (2017).
- [11] Gregory Kahn et al. “Uncertainty-aware reinforcement learning for collision avoidance”. In: *arXiv preprint arXiv:1702.01182* (2017).

- [12] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.
- [13] H Jin Kim et al. “Autonomous helicopter flight via reinforcement learning”. In: *Advances in neural information processing systems*. 2004, pp. 799–806.
- [14] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [15] Nate Kohl and Peter Stone. “Policy gradient reinforcement learning for fast quadrupedal locomotion”. In: *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. Vol. 3. IEEE. 2004, pp. 2619–2624.
- [16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [17] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [18] Piotr Mirowski et al. “Learning to Navigate in Cities Without a Map”. In: *CoRR* abs/1804.00168 (2018). arXiv: 1804.00168. URL: <http://arxiv.org/abs/1804.00168>.
- [19] Emilio Parisotto and Ruslan Salakhutdinov. “Neural Map: Structured Memory for Deep Reinforcement Learning”. In: *CoRR* abs/1702.08360 (2017). arXiv: 1702.08360. URL: <http://arxiv.org/abs/1702.08360>.
- [20] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural networks* 21.4 (2008), pp. 682–697.
- [21] Charles Richter, William Vega-Brown, and Nicholas Roy. “Bayesian learning for safe high-speed navigation in unknown environments”. In: *Robotics Research*. Springer, 2018, pp. 325–341.
- [22] Charles Richter, John Ware, and Nicholas Roy. “High-speed autonomous navigation of unknown environments using learned probabilities of collision”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 6114–6121.
- [23] Stéphane Ross et al. “Learning monocular reactive UAV control in cluttered natural environments”. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1765–1772.
- [24] Fereshteh Sadeghi and Sergey Levine. “Cad2rl: Real single-image flight without a single real image”. In: *arXiv preprint arXiv:1611.04201* (2016).
- [25] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Vol. 3. Cambridge University Press, 1999.

- [26] Patrice Y Simard, David Steinkraus, John C Platt, et al. “Best practices for convolutional neural networks applied to visual document analysis.” In: *ICDAR*. Vol. 3. 2003, pp. 958–962.
- [27] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.
- [28] Y. Tassa, N. Mansard, and E. Todorov. “Control-Limited Differential Dynamic Programming”. In: *IEEE Conference on Robotics and Automation (ICRA)*. 2014.
- [29] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.