Dynamic Obstacle Avoidance in Model-predictive Robot Planning

Jason White, Tianze Wang, David Jay, and Christian Hubicki

***Abstract*— We present a real-time motion planner that avoids multiple moving obstacles without knowing their dynamics or intentions. This method uses convex optimization to generate trajectories for linear plant models over a planning horizon (*i.e.* model-predictive control). While convex optimizations allow for fast planning, obstacle avoidance can be challenging to incorporate because Euclidean distance calculations tend to break convexity. By using a half-space convex relaxation, our planner reasons about an approximated distance-to-obstacle measure that is linear in its decision variables and preserves convexity. Further, by iteratively updating the relaxation over the planning horizon, the half-space approximation is improved, enabling nimble avoidance maneuvers. We further augment avoidance performance with a soft penalty slack-variable formulation that introduces a piecewise quadratic cost. As a proof of concept, we demonstrate the planner on double-integrator models in both single-agent and multi-agent tasks–avoiding multiple obstacles and other agents in 2D and 3D environments. We show extensions to legged locomotion by bipedally walking around obstacles in simulation using the Linear Inverted Pendulum Model (LIPM). We then present two sets of hardware experiments showing real-time obstacle avoidance with quadcopter drones: (1) avoiding a 10m/s swinging pendulum and (2) dodging a chasing drone.**

I. Introduction

Planning with avoidance is necessary for robots in real-world environments. Whether in factories, hospitals, or exploring the outdoors, robots need to avoid collisions with obstacles or other agents without advance knowledge of their intentions. Further, as robots become increasingly fast and their environments more dynamic, motion planning must be expedited to keep pace. This work presents a convex motion planning optimization capable of avoiding dynamic obstacles without knowing their intentions that solves in real time (on the order of 1kHz). The computation speed and reliable optimization convergence make the planner suitable to highly dynamic, real-world applications.

A seminal method for avoiding dynamic obstacles is to detect and avoid a collision cone as computed by the motion of "velocity obstacles" [1]. Optimal Reciprocal Collision Avoidance (ORCA) builds atop this concept to find collision-free motions. It does so by finding sets of velocities for each agent that would create collisions and disallowing them. These methods scale well to many agents and do not require explicit communication, but instead make assumptions about other agents to guarantee collision-free strategies [2]

*Toyota Research Institute provided funds to support this work.

[1]All authors are with the Department of Mechanical Engineering, Florida State University, FAMU-FSU College of Engineering, Tallahassee, FL 32310, USA. Corresponding author: `jw18dn@fsu.edu`

Supporting Video: https://youtu.be/cKKLuRdrqcQ

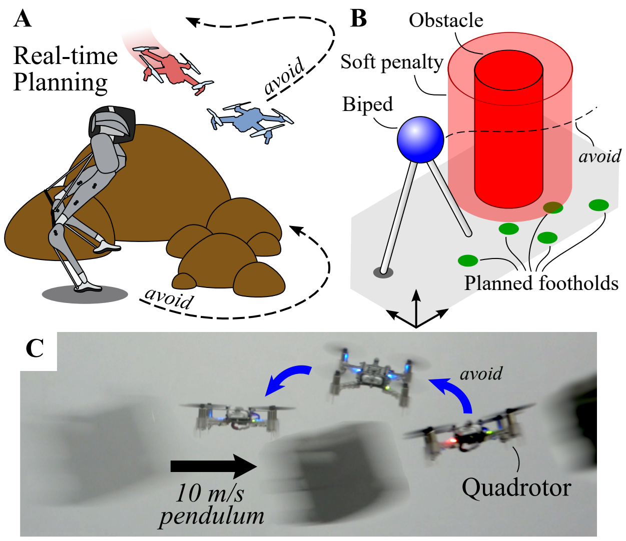

Fig. 1: **A.** Robots must quickly plan their motions to avoid obstacles and adversaries in real-world environments. **B.** Our optimization methods use linear constraints and costs to achieve reliable real-time motion planning that spans locomotion modes from flying to walking. **C.** Demonstration of a quadrotor evading a 10m/s swinging pendulum. Note: drone positions were adjusted for image clarity, see supplementary video for unadjusted footage.

which have since been unified across agent plant dynamics [3]. However, these guarantees are less reliable under the practical actuation limitations of robots. Other methods explicitly use mutual communication to update collision-free trajectories [4]. Further, safety-focused control methods like barrier functions [5] use linear constraints to divert agents away from failure modes such as obstacles without explicitly treating them as agents.

Optimization approaches aim to use real-time motion planning (*i.e.* Model Predictive Control, or MPC) to flexibly generate control on-the-fly for quadrotors [6], fixed-wing aircraft [7], quadrupeds [8], bipeds [9]. Further, these methods can be more robust to dynamical assumptions about obstacles. MPC has the additional benefit of minimizing a cost function while respecting constraints, (*e.g.* physical limits, and obstacles [10]). To be effective, the motion planners within MPC must be sufficiently fast and reliable to react in real time, and thus turn to fast and reliable convex optimizations [11]. While MPC typically assumes convex linear constraints, the Euclidean distance constraints inherent to obstacle avoidance are nonlinear and thus impede convexity. Convex relaxations have formulated convex SOCPs [12] which solve at 8-30Hz, and fast nonlinear solvers like

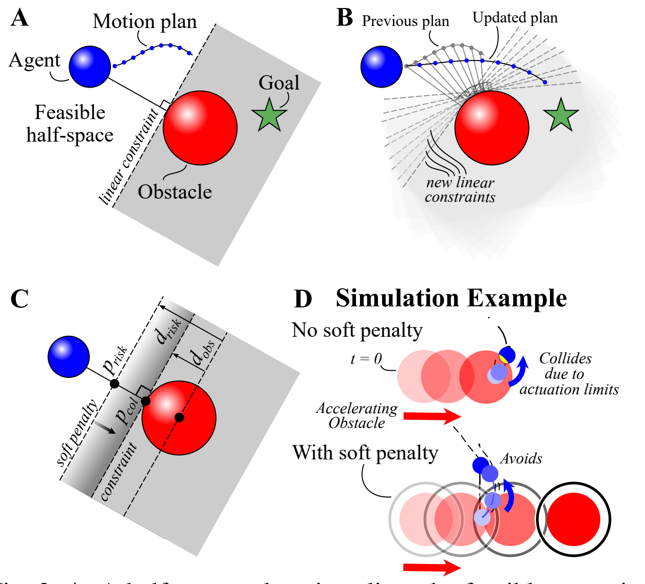

Fig. 2: **A.** A half-space relaxation slices the feasible space in half with the plane tangent to the closest obstacle point. **B.** Half-spaces are recut using the previous planned trajectory when computing the closest obstacle point, iteratively improving the half-space approximation. **C.** We add a quadratic cost once the agent crosses a threshold close to the half-space cut. **D.** In simulation, this piecewise cost improves avoidance when the agent has physical limits.

SQP [13] and embedded solvers like PANOC [14] and have been practically successful in achieving kHz control rates.

We seek an MPC formulation that leverages well-developed fast convex solvers, specifically for quadratic programs (QPs), that reliably avoid dynamic obstacles and adversaries. QP solvers are well-developed and known to have kHz computation rates [15]. We leverage a half-space convex relaxation of the obstacle-free space [16] and iteratively update that approximation over the MPC planning horizon. Using a slack-variable formulation, we add a penalty term that serves as a fast-solving piecewise quadratic cost. In this paper we show that this real-time approach to dynamic obstacle avoidance can nimbly avoid dynamic obstacles and adversaries in a manner that:

1) Scales to multiple obstacles and 3D
2) Handles non-cooperative agents and adversaries
3) Extends to varied locomotion modes (e.g. bipeds)
4) Is effective on hardware in real-time.

The overall contributions of this paper are:

1) A soft piecewise constraint that penalizes the cost function when an agent passes an obstacle's "at risk" distance–as computed by the half-space relaxation
2) A numerical and physical parameter study that quantifies the reliability of the soft constraint formulation
3) A validation of the method on multiple simulated systems with varying dynamics and on quadrotor hardware in dynamic avoidance tasks.

II. Methods

Here we present a method of dynamic obstacle avoidance using a convex MPC motion planning framework. We show that a half space representation of obstacles is sufficient for practical real time avoidance, given the avoidance is treated as a penalty rather than a hard constraint. Using the previous motion plan, piecewise constrained slack variables penalize the agent when it passes an obstacle distance threshold.

A. Conventions

The following notation is used throughout the paper: A right subscript of k refers to node k along the trajectory where $k \in 1,2,...,N$ and N is the number of nodes. $\mathbb{R}, \mathbb{R}^n, \mathbb{R}^{n\times m}$ represent the space of real numbers, vectors with length n, and matrices with n rows and m columns. Scalar values are lowercase and italicized letters (e.g. $x \in \mathbb{R}$). Bold, lowercase letters represent vectors (e.g. $\mathbf{p}_k \in \mathbb{R}^n$) while bold, uppercase letters represent matrices (e.g. $\mathbf{P} \in \mathbb{R}^{n\times m}$). Variables $\mathbf{p}_k$, $\mathbf{v}_k$, $\mathbf{u}_k$, δ_k are positions, velocities, control inputs, and slack variables with the robot state vector defined as $\mathbf{q}_k = [\mathbf{p}_k \ \mathbf{v}_k]^T$ and its derivative $\dot{\mathbf{q}}_k = [\dot{\mathbf{p}}_k \ \dot{\mathbf{v}}_k]^T$. The double integrator model is represented in three dimensions $\mathbb{R}^3$, for (x,y,z), and the LIPM in two dimensions $\mathbb{R}^2$, for (x,y). Right subscripts with a comma ($\mathbf{q}_{prev,k+1}$) indicate a description of the variable, followed by the node number. A star superscript indicates state variable k along the trajectory with $\mathbf{p}^*_{\text{agent},k}$ being the agents most recent planned trajectory and $\mathbf{p}^*_{\text{obst},k}$ being the estimated obstacle trajectory. $\mathbf{W}_\star$ denote positive semi-definite symmetric weighting matrices for cost category, $\star$.

B. Assumptions

We assume the initial positions $\mathbf{p}_1$ and velocities $\mathbf{v}_1$ of both the agent $\mathbf{q}_{\text{agent}}$ and obstacle $\mathbf{q}_{\text{obst}}$ are known. As is common in most MPC implementations, we assume linear plant dynamics. Although the obstacles are simulated using double integrator dynamics, our avoidance constraint formulation Sec. II-D uses only the obstacle's initial states ($\mathbf{q}_{\text{obst},1}$) with a simple single integrator assumption of the dynamics. Specifically, the obstacle's future motion is estimated using only its current states $\mathbf{q}_{\text{obst},1}$ integrated forward in time, ΔT, per:

$$\mathbf{p}^*_{\text{obst},k+1} = \mathbf{p}_{\text{obst},1} + \mathbf{v}_{\text{obst},1} \ \mathbf{k} \ dt, \quad \mathbf{k} = \{1,2,...,N-1\} \quad (1)$$

Despite these simplifying model assumptions, we will demonstrate its effective avoidance behaviors even when implemented in crowded multiagent scenarios (Sec. III-D.1). In the presented multi-agent and multi-adversary scenarios, there is no communication of plans or intent across agents–and adversaries' actual motion is generated using second-order system dynamics with a PD control law to model chasing behavior.

C. Model Predictive Control

MPC is a common method of using online optimizations to track desired trajectories but can also be used for online motion generation with loose (or non-existent) tracking requirements. We use it to generate the robot state vectors $\mathbf{q}_k$ and associated control inputs $\mathbf{u}_k$ while avoiding obstacles–facilitated in part by including a slack variable δ_k.

As is typical in MPC, we facilitate model prediction in our equality constraints using an Explicit Euler integration of our system dynamics. At each control loop, we constrain the measured states to equal the initial trajectory states. The optimization uses a multi-part quadratic cost, and linear constraints per:

$$\begin{aligned}
\min_{\mathbf{q}_k,\mathbf{u}_k,\delta_k} \quad & \underbrace{J(\mathbf{q},\mathbf{u},\mathbf{s})}_{\text{Task}} + \sum_{k=1}^{N} \underbrace{||\delta_k||^2_{\mathbf{W}_{\text{avoid}}}}_{\text{Avoidance}} \\
\text{s.t.} \quad & \dot{\mathbf{q}}_k = \mathbf{A}\mathbf{q}_k + \mathbf{B}\mathbf{u}_k && \text{(Dynamics)} \\
& \mathbf{q}_{k+1} = \mathbf{q}_k + \dot{\mathbf{q}}_k dt && \text{(Model Prediction)} \\
& f(\mathbf{q}_k, \delta_k) \le 0 && \text{(Avoidance)} \\
& h(\mathbf{q}_k, \mathbf{u}_k) \le 0 && \text{(Task Constraints)}
\end{aligned}$$

and is detailed in later sections, where the cost function $J(\mathbf{q},\mathbf{u},\mathbf{s})$ has a corresponding weighting matrix, $\mathbf{W}$, and is specific to each task and model (J for brevity).

D. Convex Obstacle Avoidance

Determining an avoidance strategy for even a simple circular object is inherently a nonlinear problem by the nature of the Euclidean distance formula. To avoid this nonlinearity, we instead relax the Euclidean distance using a half-space representation (or relaxation) of the distance constraint. We draw a plane that bisects the space into feasible and infeasible regions by finding the obstacle point nearest to the agent and defining a cutting plane tangent to it Fig. 2A. Distance to this plane is a linear function of our design variables, and thus can be formulated as convex linear constraints.

These half-space relaxations, while convex, have the potential to overly limit the mobility of the agent. However, we can improve this approximation by iteratively updating the planar cuts along the planned trajectory by using the previously generated motion plans as seen in Fig. 2B. To accomplish this we find the distance between a point of interest on the agent and obstacle via:

$$\mathbf{d}_k = \mathbf{p}^*_{\text{agent},k} - \mathbf{p}^*_{\text{obst},k} \tag{2}$$

Since the vector is from the agent and obstacles point of interest, we need to scale this vector to find the distance to a collision. For simplicity this paper only considers the agent and obstacles to be spherical, however these obstacle can take on different shapes. The constraint then becomes:

$$\mathbf{d}_k \cdot (\tilde{\mathbf{d}}_k - \mathbf{p}_k) \le 0. \tag{3}$$

where $\tilde{\mathbf{d}}_k$ is the offset necessary to represent the obstacle and agent width along $\mathbf{d}_k$. This is calculated as $\tilde{\mathbf{d}}_k = (d_{\text{agent}} + d_{\text{obst}})\hat{\mathbf{d}}_k$ where d_{agent} and d_{obst} are simply the respective radii of the obstacle and agent for spherical objects, and $\hat{\mathbf{d}}_k$ is the unit vector of $\mathbf{d}_k$.

E. Soft Penalty

This method alone only produces a rough estimate of the obstacle future states, and thus commonly violates constraints leading to infeasible optimizations. To resolve this we define an additional "at risk of collision" distance d_{risk} which acts a buffer to penalize the optimization when the agent enters this space. We develop a second planar cut, parallel to the first, but include an additional design variable in the equation Fig. 2C. This design variable (δ_k) is a slack variable, and will penalize the optimization based on the distance between the obstacle and agent. We bound this penalization by including a second constraint on the slack variable. This peicewise formulation ensures there is no penalty outside of soft penalty line, but escalates quickly after crossing the soft penalty threshold. These avoidance constraints for $f(\mathbf{q}_k, \delta_k)$ are written as such:

$$\begin{aligned}
-\delta_k \le 0 \quad & \text{(No-penalty Region)} \\
\mathbf{d}_k \cdot (\tilde{\mathbf{d}}_k - \mathbf{p}_k) - \delta_k \le 0 \quad & \text{(Penalty Region)}
\end{aligned} \tag{4}$$

where $\tilde{\mathbf{d}}_k = (d_{\text{agent}} + d_{\text{obst}} + d_{\text{risk}})\hat{\mathbf{d}}_k$, and the slack variable δ_k is included in the cost function to enforce the penalties associated with the agent encroaching on the obstacle's defined threshold,

$$J_{\text{avoid}}(\delta) = \sum_{k=1}^{N} ||\delta_k||^2_{\mathbf{W}_{\text{avoid}}}. \tag{5}$$

This general avoidance formulation scales well computationally when expanded to many obstacles so long as the plant dynamics are approximated as linear–examples of which are detailed in the sections that follow.

F. Double Integrator Model Example

We introduce the state space model for a double integrator (Eq. 6) as a minimalist representation of mechanical locomotor dynamics. We use it to both simulate the avoidance method, and to generate trajectories online for our quadrotor hardware. The dynamic model we use allows each Cartesian direction to be controlled, resulting in the state space model:

$$\dot{\mathbf{q}}_k = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{q}_k + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{u}_k. \tag{6}$$

In addition to the constraints presented in the previous section, the model has both cartesian bounds $h(\mathbf{q}_k)_1 = \mathbf{p}_{min} \le \mathbf{p}_k \le \mathbf{p}_{max}$ and control input bounds $h(\mathbf{u}_k)_2 = \mathbf{u}_{min} \le \mathbf{u}_k \le \mathbf{u}_{max}$. The cost function for this model and the LIPM (presented in the next section), have significant overlap and thus are detailed in Sec. II-H.

G. Linear Inverted Pendulum Example

The Linear Inverted Pendulum Model (LIPM) has been widely used in control design for legged robots, particularly motion generation [17]. It has been effective in capturing the essential dynamics of bipedal walking with its simple linear formulation. The LIPM simplifies bipedal dynamics by assuming a point center of mass (CoM) for the main body with two massless legs. By assuming small vertical

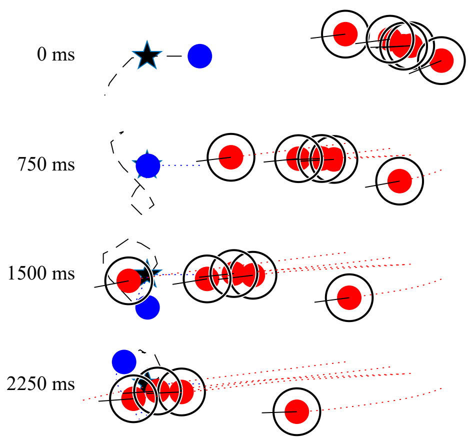

Fig. 3: Simulated results of 2D uncoordinated waypoint tracking single agent (blue) with multi-adversary (red) avoidance. Despite the half-space constraints, the agent planned figure-eight looping maneuvers around adversaries. The four images show the tracking and avoidance trajectory through time (black) and the obstacle's traveled paths (red) over a 2.25 second period.

deviations of the center of mass, the plant dynamics become linear. We define the the center of mass motion and the two feet in x, y Cartesian space as shown in Fig. 8A. The dynamics of the LIPM are:

$$\dot{\mathbf{q}}_k = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{M} & \mathbf{0} \end{bmatrix} \mathbf{q}_k - \begin{bmatrix} \mathbf{0} \\ \mathbf{M} \end{bmatrix} \mathbf{u}_k; \quad \mathbf{M} = \begin{bmatrix} \frac{g}{z_0} & 0 \\ 0 & \frac{g}{z_0} \end{bmatrix} \tag{7}$$

where g is the gravity term, z_0 is the walking height, $\mathbf{q}_k$ is the predicted center of mass positions and velocities $\mathbf{q}_k = [\mathbf{p}_k, \ \mathbf{v}_k]^T$, and $\mathbf{u}_k$ is the predicted center of pressure. For the LIPM, $\mathbf{u}_k$ in Eq. 7 is equivalent to the foot position since the LIPM assumes massless legs, a constant robot height z_0, and point feet. Generally, the center of pressure position can be any point within a convex region around the foot position. We formulate LIPM motion planning as a convex MPC problem using reachability constraints and step length deviation cost as seen in [18], [19].

1) Reachability Constraints: A reachability constraint is included to ensure the planned foot locations are within the kinematic limits of the leg and is defined as for $h(\mathbf{q}_k, \mathbf{u}_k)$ in the following linear form:

$$\mathbf{s}_{\text{min}} \leq (\mathbf{u}_k - \mathbf{p}_k) \leq \mathbf{s}_{\text{max}} \tag{8}$$

where $\mathbf{s}_{\text{min}}, \mathbf{s}_{\text{max}}$ represent the maximum and minimum distance the feet can be away from the robot body.

2) Step Length Cost: This cost function term is specific to the LIPM, and minimizes the difference between the predicted step length $\mathbf{s}$ and the reference step length $\mathbf{s}_{\text{ref}}$. This drives the robot foot to the desired position from the current foot placement, and is formulated as such:

$$J_{\text{step}}(\mathbf{s}) = \sum_{k=1}^{N_s} ||\mathbf{s}_k - \mathbf{s}_{\text{ref},k}||^2_{\mathbf{W}_{\text{step}}} \tag{9}$$

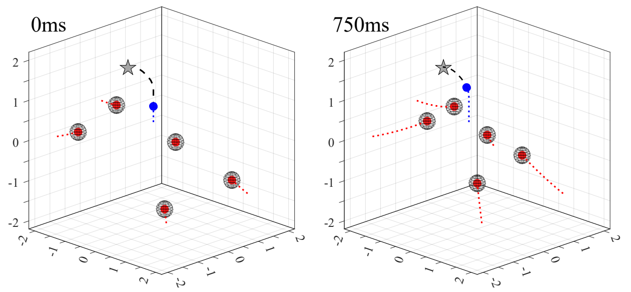

Fig. 4: Simulated results of 3D uncoordinated waypoint tracking single agent (blue) with multi-adversary avoidance. Left and right images are sampled 0.75s apart.

where N_s is the number of robot steps predicted, and the number of MPC prediction steps is found using $N = N_s \frac{t_{\text{step}}}{dt}$. The frontal reference $s_{\text{ref},x}$ (or desired) step length is computed per:

$$s_{\text{ref},x} = v_{\text{ref},x} \ t_{\text{step}} \tag{10}$$

using the x directional walking speed $v_{\text{ref},x}$ and stepping time t_{step}. The sagittal step length $s_{\text{ref},y}$ is found using a nominal distance between the foot and robot CoM.

Based on the foot placement, we can compute the desired sagittal plane center of pressure reference trajectory $\mathbf{u}_{\text{ref},x} \in \mathbb{R}^N$ for the J_{step} cost function term per:

$$\mathbf{u}_{ref,x} = \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \dots \\ \dots \\ \mathbf{1} \end{bmatrix} p_{\text{foot},x,1} + \begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots \\ \mathbf{1} & \mathbf{1} & \dots & \mathbf{1} \end{bmatrix} \begin{bmatrix} s_{x,1} \\ s_{x,2} \\ s_{x,3} \\ \dots \\ s_{x,N_s} \end{bmatrix} \tag{11}$$

where $\mathbf{0}$ and $\mathbf{1}$ are column vectors with length $N_r = \frac{t_{\text{step}}}{dt}$. The x directional current foot placement is represented as $p_{\text{foot},x,1}$ and x directional predicted step length as s_x. The frontal plane center of pressure reference $(\mathbf{u}_{\text{ref},y})$ is found by substituting $p_{\text{foot},y,1}$ and $s_{x,1:N_s}$ in Eq. 11.

H. Cost Function

Although the weight matrix values vary between models, components of the task cost function J_{task} can be generalized and are independent of the model. One common component in our examples minimizes the difference between the predicted states and the desired target state,

$$J_{\text{target}}(\mathbf{q}) = \sum_{k=1}^{N} ||\mathbf{q}_k - \mathbf{q}_{ref,k}||^2_{\mathbf{W}_{\text{target}}}, \tag{12}$$

where $\mathbf{q}_{ref}$ is the target state, with weighting matrix $\mathbf{W}_{\text{target}}$. For bipedal robots, the desired states $\mathbf{q}_{\text{ref}}$ are generated based on a commanded input (*e.g.* stepping in place or walking forward with certain velocity).

Another common cost component minimizes the difference between control inputs and reference control inputs,

$$J_{\text{effort}}(\mathbf{u}) = \sum_{k=1}^{N} ||\mathbf{u}_k - \mathbf{u}_{ref,k}||^2_{\mathbf{W}_{\text{effort}}}. \tag{13}$$

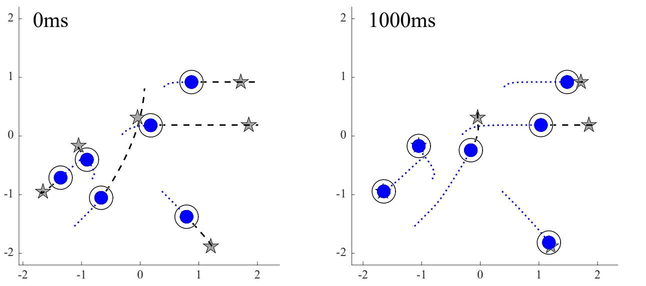

Fig. 5: Simulated results of 2D uncoordinated multi-agent avoidance and waypoint tracking. Black dashed lines are motion plans. Left and right images sampled 1 second apart.

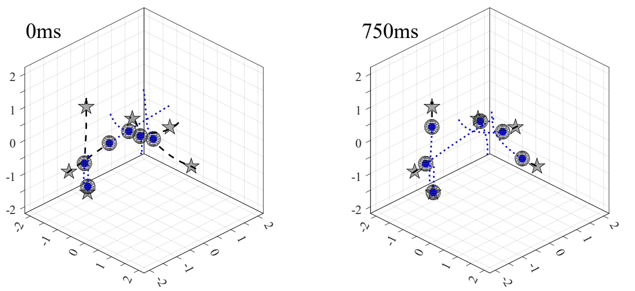

Fig. 6: Simulated results of 3D uncoordinated multi-agent avoidance and waypoint tracking. Left image shows the initial plan to waypoints with black dashed lines, and the right shows 0.75 second into the simulation.

For the LIPM, this refers to minimizing the difference between the predicted control inputs $\mathbf{u}$ and the foot position reference $\mathbf{u}_{\text{ref}}$ – encoding a preferred leg spread. In double-integrator demonstrations the cost function is $J = J_{\text{target}} + J_{\text{effort}}$, while the LIPM demonstrations include the stepping cost, $J = J_{\text{target}} + J_{\text{effort}} + J_{\text{step}}$.

III. Numerical Results

A. Simulated Environment

Simulations were run in MATLAB 2019b using the quadratic programming solver `quadprog` from the optimization toolbox. The constraints and cost function are pre-generated prior to the control loop using the `MatlabFunction` code generator and symbolic toolbox. After each elapsed sample period, the states are sampled, the planning optimization solved, and the planned control inputs are applied to the system dynamics–which are subsequently integrated with MATLAB's `ode45` medium-order differential equation solver.

A consistent 1.5sec time horizon is used with a 50ms sample time. This sample time is longer than necessary given a typical optimization required only $\sim$7ms of computation time. Recent testing has further decreased this time to $\sim$2ms (500Hz) using the OSQP solver [15].

B. Dynamic Obstacle

Early testing of the avoidance setup included an obstacle moving at a constant acceleration across the agent's desired target point. The agent then plans an avoidance maneuver with a trajectory leading back to the desired state. Originally, we simply used a half-plane constraint to represent the obstacle Fig. 2A. However, linear estimations of the dynamics and discrete time intervals commonly resulted in avoidance constraint violations and thus infeasibilities. We found adding the soft avoidance constraints to be an effective method of mitigating optimization failures–drastically improving obstacle avoidance. Specifically, it allowed previous optimizations to solve reliably, as depicted in Fig. 2D. We further explore similar single-obstacle scenarios on drone hardware in Sec. IV.

C. Adversarial Pursuit

We expand the previous scenario by altering control of the obstacle(s) to be adversarial. Using a basic PD control law, the adversary(s) (or obstacle(s)) track and chase the agent. The agent is tasked to go to a target point which is randomly moved to a new position every 4 seconds. The dynamics for both systems are nearly identical (Eq. 6), with the adversary(s) having a small damping term to reduce overshoot when tracking the agent. Again, these are non-cooperative entities, and the agent has no information about the adversary's intent other than its current position and velocity. We find even when we limit the agent's velocity to be substantially lower than the adversary's velocity, the agent successfully avoids the adversary.

1) 2D Multi-obstacle: Here we increase the complexity by avoiding 5 adversaries, each with randomized gains and starting positions Fig. 3. In some situations it would appear the agent (in blue) was trapped, but this simple heuristic method would find routes which encroached on the adversary's (in red) soft constraints (black circle). Sec. III-E quantifies the performance reliability with up to 10 adversaries, but we were able to demonstrate successful evasion with up to 15, the largest number tested.

2) 3D Simulation: We further extend the method to three dimensions and find similar results Fig. 4. Using constraint planes to navigate around obstacles we demonstrate successful scaling to 3D in both performance and tractable computation.

D. Multiple Agents (Go Home)

We now demonstrate collision avoidance among multiple MPC agents that are neither adversarial nor cooperative. We simulate multiple agents, independently optimizing their motion plans to reach target points and assess their performance when their planned paths conflict.

1) 2D Multiple Agents: Here we model 6 systems, all planning agents, in a 2D environment and randomize new target points every 4 seconds. We again find similar results Fig. 5 to the obstacle avoidance, where this heuristic setup is robust to a dynamic environments, and the agents do not collide with each other. The setup presented here uses individual QP's solved for each agent's trajectory.

A common mode of failure is the "who goes first?" conundrum–where two agents moving in parallel need to cross each other such that one must speed up or slow down (*i.e.* a stalemate). This was particularly problematic in earlier

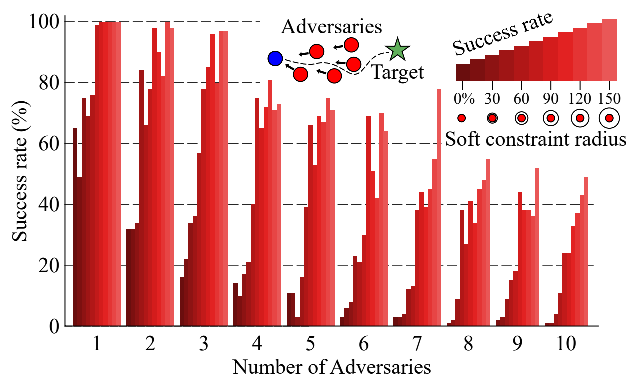

Fig. 7: Reliability testing of multiagent avoidance in a tight corridor as a function of soft constraint radius.

formulations that did not iteratively update half-space cuts along previously solved trajectories. Including the iterative half-space update anecdotally reduced the incidence of these stalemates. Although our multi-agent results are independent optimizations, we could generate cooperative swarming at the cost of longer computation times (*i.e.* multi-agent MPC) by combining all planning operations into a single coupled optimization.

2) 3D Multiple Agents: We further extend the method to three dimensions and find similar results as seen in Fig. 6B, again demonstrating the proposed method scales effectively to 3D for multiple non-cooperative agents.

E. Reliability Results

To understand the effect of our soft constraint, we simulate the 2D system starting with no soft penalty, and ending with a soft penalty 1.5 times the $\mathbf{d}_{obs}$ for 1 through 10 obstacles.

$$\begin{aligned} 0 &\geq \mathbf{d}_k \cdot (\tilde{\mathbf{d}}_k - \mathbf{p}_k) - \delta_k && \text{(Soft Constraint)} \\ \tilde{\mathbf{d}}_k &= (d_{\text{agent}} + d_{\text{obst}} + d_{\text{risk}})\hat{\mathbf{d}}_k \\ d_{\text{risk}} &= \mathbf{j}\ d_{\text{obst}} && \mathbf{j} = \{0, 0.3, ..., 1.5\} \end{aligned}$$

Starting at a position of $\mathbf{p}_{\text{agent},1} = [4,0]^T$, the agent is tasked to travel to $\mathbf{v}_{\text{agent},1} = [0,0]^T$, with adversarial obstacles placed in its path. Initial obstacle locations are selected at random within a range of $1 \leq x \leq 3$ and $-0.5 \leq y \leq 0.5$. Each obstacle uses a PD control law with randomized gains to "chase" the agent. Each simulation runs until the agent either reaches the goal with a velocity near zero, or collides with an obstacle. The corresponding success rates out of 100 trials are presented in Fig. 7.

F. Legged Locomotion

In this section, we will show that the obstacle avoidance algorithm can be easily adapted for other robotics applications like legged robot motion generation.

Here we show the LIPM-MPC simulation result Fig. 8A. The robot stepping time T_{step} is 0.5s with a sample time ΔT of 0.1s. The motion planner predicts $N_s = 4$ steps into the future and we command the robot to simply walk forward. The simulated results without obstacle avoidance Fig. 8B and with obstacle avoidance Fig. 8C show that the avoidance algorithm can guide the robot to walk around the obstacle safely. Further, we can tune the MPC soft penalty terms and add more points to the robot for collision detection (*e.g.* robot feet, knees, etc.) to extend the LIPM-MPC capabilities for more-complex geometries and dynamic environments.

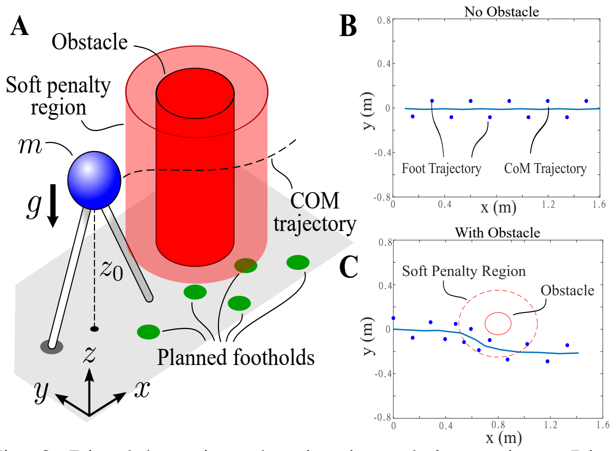

Fig. 8: Bipedal motion planning in real-time using a Linear Inverted Pendulum Model (LIPM). **A.** Diagram for the numerical experimental setup. **B.** An MPC-generated motion without an obstacle. **C.** Real-time bipedal obstacle avoidance using the presented motion planner.

IV. Hardware Experiments

A. Experimental Setup

1) Hardware Description: Hardware experiments were carried out using a BitCraze quadrotor. The BitCraze quadrotor CrazyFlie uses brushed DC motors and are controlled from a Dell XPS 8700 with an Intel i7-4790 processor and 14GB of RAM base computer using a 2.4Ghz radio USB dongle. For tracking the quadrotor's state, we use a Vicon motion capture system. Our Vicon setup uses eight infared cameras to track the position of reflective markers, and measures drone states with a sub-millimeter precision at an approximate sampling rate of 150Hz.

2) Software: BitCraze uses open-source code which runs a low-level control loop at 1kHz and has been extended/modified by a number of other universities. The University of Southern California developed a software package called "Crazyswarm" [20] that uses a ROS node to control a swarm of up to 50 drones. We use this code base because of its motion capture integration, simulation environment, and user friendly Python-based firmware. Of the available motor level controllers (PID, INDI, or Mellinger) from Crazyswarm, we use the Mellinger controller [21]. It allows for steep roll/pitch angles, and develops minimum snap trajectories in real time using a sequence of desired Cartesian positions $[x\ y\ z]$, yaw angle (ψ), and their subsequent derivatives. We include our avoidance method in the repository by calling MATLAB from Python, and writing a `.csv` file from the trajectory points planned by MPC. The main execution control loop runs at 20Hz using points from the most recent trajectory.

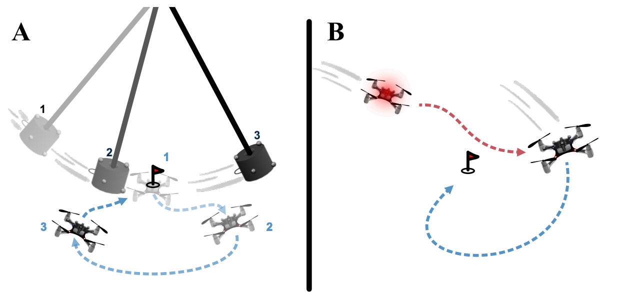

Fig. 9: In our experiments, the quadrotor must minimize its distance to a waypoint while **A.** avoid a swinging pendulum and **B.** avoiding a chasing adversary quadrotor controlled by a PD chase law.

B. Systematic Experiment: Pendulum Avoidance

To test the effectiveness of the "at risk" soft distance constraint, we incrementally changed the $\mathbf{p}_{risk}$ value for the quadrotor, and proceeded to swing a pendulum at it (3 times for each of the 7 radii), as seen in Fig. 9A. The 0.2kg pendulum bob was encased in foam, and hung by fishing line 3.2m below the ceiling. To begin the test, the pendulum is set at a fixed height on a post and held using a pin-release mechanism. The drone is placed in the motion capture volume and commanded to hover 0.3m above the origin so that the pendulum is sure to strike it. For each experiment, the pendulum was released from the same height and the drone was commanded to hold the same desired state. As seen in Fig. 10, a $d_{risk} \geq$ 15cm worked every time–even with the pendulum released from the highest possible height, **reaching speeds greater than 10m/s**.

C. Demonstration: Multi Pendulum Avoidance

We further extend the pendulum avoidance experiment by introducing 3 pendulums swinging perpendicular to the drone's desired way point. In the experiment detailed in Fig. 12, the first pendulum swings directly into the drone's path at the start of the experiment. As demonstration of dynamic avoidance, the drone reacts to avoid the incoming obstacle and then slows down to avoid hitting the second pendulum before proceeding to its final target position.

D. Demonstration: Adversarial Pursuit

We validate the numerical results from Section III by mimicking the adversary setup. As seen in Fig. 11B, two CrazyFlie quadrotors are placed apart from each other at random positions within the motion capture field. The agent uses the MPC avoidance method, the adversary chases the agent position using a PD controller, and both use a Mellinger controller for low-level control.

1) 2D Adversary Avoidance: Our initial adversarial experiments constrained the avoidance to two dimensions. Representative experimental data can be seen in Fig. 11B where the adversary approaches the agent, causing the agent to plan a trajectory around the adversary, temporarily leaving its target point to dodge.

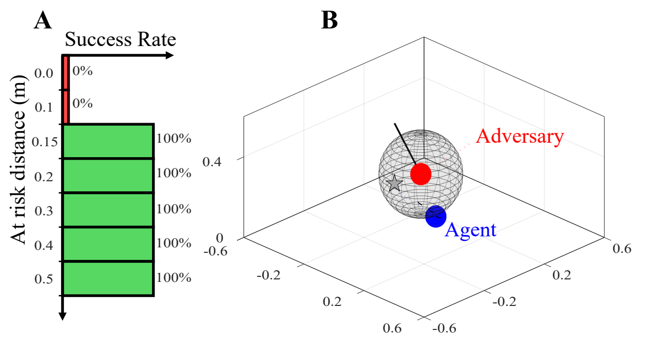

Fig. 10: Pendulum experiment results. **A.** Soft penalty distance, d_{risk} and the avoidance success rate for 3 pendulum swing attempts. **B.** Example trajectory data from pendulum test.

2) 3D Adversary Avoidance: We extended our experiments into three dimensions as seen in the experimental results in Fig. 11C. Similar to the 2D adversary avoidance tests, the agent was able to avoid the chasing adversary. One notable effect was the down-wash disturbance when one drone passed below the other, which would cause instability in the PD-controlled chase drone.

V. Discussion

A. Avoidance Behavior

There was a clear correlation between the slack variable weight $\mathbf{W}_{\text{avoid}}$, the soft constraint distance d_{risk}, and the avoidance performance. When the weight was low, the agent would favor passing the soft constraint, commonly resulting in a collision. Alternatively, a small soft constraint distance with a very large weight resulted in more reliable obstacle avoidance. Overall we find that making the slack variable weight significantly larger than the other cost function weights to be the most effective method of enforcing avoidance.

Although methods for automatically tuning these parameters is beyond the scope of this paper, we believe that developing these values depends on both the obstacle's/agent's dynamically constrained maneuverability relative to each other and the obstacle's/agent's size. We realize the values at each node could be chosen using a more methodical manner, but the results presented here simply use a scalar value across all nodes for the soft constraint penalty and soft constraint distance. Despite this simplification, the agents, both simulated and physical, are able to successfully avoid dynamic obstacles, which we believe speaks to the efficacy of the proposed half space method with soft constraints.

VI. Conclusion

We presented a motion planning method that allows for cross platform real-time planning while avoiding multiple uncooperative and adversarial obstacles in 2D and 3D environments. Even when the obstacles are adversarial or move at high speeds, the agent safely navigates around them in both simulation and hardware. Were further applied the avoidance

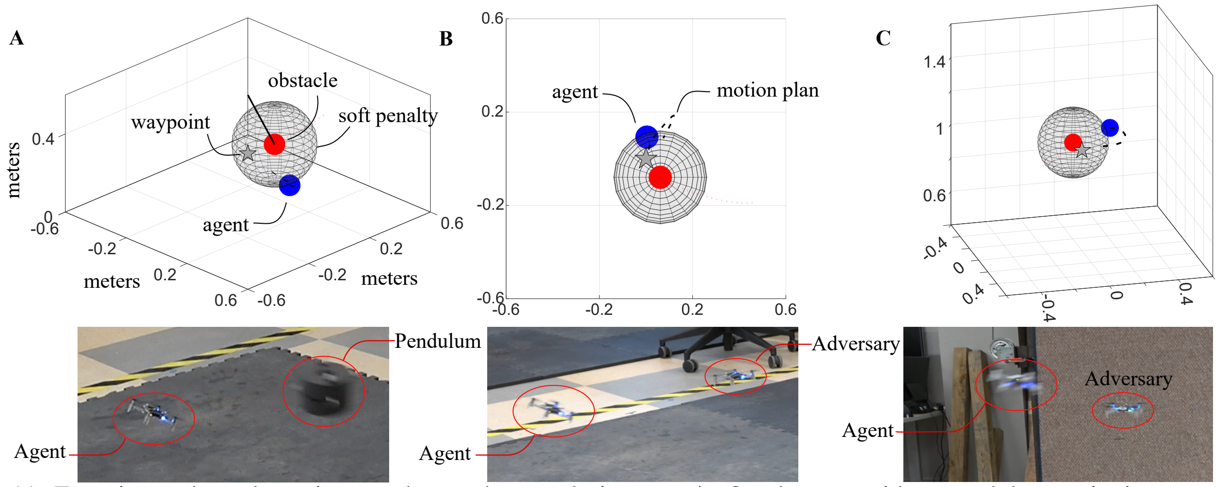

Fig. 11: Experimental results trajectory data and example images. **A.** Quadrotor avoids a pendulum swinging at speeds up to 10m/s. **B.** The quadrotor is chased by an adversary drone and must avoid while staying within a 2D horizontal plane. **C.** Quadrotor adversary chase in 3D. See supplementary video for real-time footage.

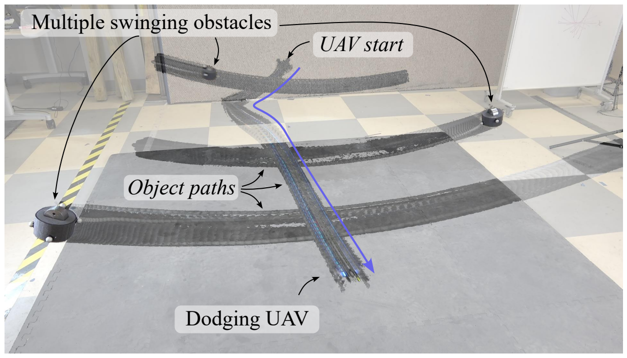

Fig. 12: Stroboscopic image of a quadrotor UAV avoiding multiple swinging obstacles in its path.

formulation to a bipedal walking model avoiding a stationary obstacle - demonstrating applicability to varied locomotion modes. We validated these methods on a quadrotor, tasking it to (1) avoid a 10m/s pendulum swing and (2) avoid a chasing quadrotor adversary.

References

[1] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[2] J. V. D. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.

[3] D. Bareiss and J. Van Den Berg, "Generalized reciprocal collision avoidance," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, 2015.

[4] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6753–6760.

[5] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[6] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 773–11 780, 2014.

[7] T. J. Stastny, A. Dash, and R. Siegwart, "Nonlinear mpc for fixed-wing uav trajectory tracking: Implementation and flight experiments," in *AIAA guidance, navigation, and control conference*, 2017, p. 1512.

[8] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[9] M. J. Powell, E. A. Cousineau, and A. D. Ames, "Model predictive control of underactuated bipedal robotic walking," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5121–5126.

[10] H. Cheng, Q. Zhu, Z. Liu, T. Xu, and L. Lin, "Decentralized navigation of multiple agents based on orca and model predictive control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3446–3451.

[11] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[12] M. Szmuk, C. A. Pascucci, D. Dueri, and B. Açıkmeşe, "Convexification and real-time on-board optimization for agile quad-rotor maneuvering and obstacle avoidance," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4862–4868.

[13] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Nonlinear model predictive control for multi-micro aerial vehicle robust collision avoidance," *arXiv preprint arXiv:1703.01164*, 2017.

[14] A. Sathya, P. Sopasakis, R. Van Parys, A. Themelis, G. Pipeleers, and P. Patrinos, "Embedded nonlinear model predictive control for obstacle avoidance using panoc," in *2018 European control conference (ECC)*. IEEE, 2018, pp. 1523–1528.

[15] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[16] Y. Chen, S. Huang, and R. Fitch, "Active slam for mobile robots with area coverage and obstacle avoidance," *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 3, pp. 1182–1192, 2020.

[17] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, "The 3d linear inverted pendulum mode: a simple modeling for a biped walking pattern generation," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2001, pp. 239–246.

[18] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, and M. Diehl, "Online walking motion generation with automatic footstep placement," *Advanced Robotics*, vol. 24, no. 5-6, pp. 719–737, 2010.

[19] K. Wang, H. Fei, and P. Kormushev, "Fast online optimization for terrain-blind bipedal robot walking with a decoupled actuated slip model," 2021.

[20] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3299–3304.

[21] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.