2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Collision Detection and Avoidance for Black Box Multi-Robot Navigation

Sara Ayoubi[1] , Ilija Hadžić[2], Lou Salaün[1], and Antonio Massaro[1]

Abstract— **To date, commercial industrial robots only provide multi-robot coordination for their own fleet of robots and treat robots from other vendors as general obstacles. The ability to enable robots from different vendors to co-exist in the same space is crucial to prevent vendor lock-in. We present the first decentralized system that achieves coordination between a heterogeneous fleet of black box robots for which the internals of the navigation stack are presumed unmodifiable. Our system, which we call CODAK, achieves the coordination by relying on minimum set of interfaces that are commonly available on most industrial and service robots. For each robot, CODAK uses a trained recurrent neural network to anticipate collisions from externally observable metrics. Anticipated collisions are avoided using a simple, but yet effective, concurrency control scheme. We run a series of experiments in simulation and with real robots to demonstrate CODAK's ability to enable safe navigation in different environments. We also experimentally compare CODAK with previously published white-box solutions to evaluate the penalty of black-box constraint.**

I. Introduction

We consider the problem of multi-robot navigation in an environment where robots come from different vendors, and co-exist and navigate around the free space with no guide strips, roadways, rails, or other motion-constraining fixtures. This, so called, heterogeneous multi-robot navigation has significant practical value, because it reduces the barrier of entry for the operators and eliminates vendor lock-in.

Most commercial industrial robots use a closed-system (black box) navigation stack capable of planning the path towards the goal and navigating around obstacles based on a cost map constructed out of provisioned or sensed obstacles. They typically expose an interface for monitoring and tasking, but internals of the navigation stack are kept proprietary. Multi-robot navigation is typically not designed to inter-operate across robots from different vendors.

We study the problem of how to enable a heterogeneous fleet of autonomous ground robots to simultaneously navigate a shared environment and coordinate their actions, relying only on restricted interfaces, specifically issuing the goal, pausing the robot, and retrieving the location, plan, and velocity information from the robot. This restriction limits the coordination module in what it can do to control the navigation (e.g., it cannot directly control the velocity or divert the robot from the planned path). In our system, which we call CODAK, short for COllision Detection and Avoidance for blacK box robots, each robot is paired with an external software agent that communicates with the robot and other agents, makes no assumption about the inner workings of the robots, and controls the robot over a well-defined and restricted interface. CODAK works by learning a stochastic motion prediction model of all participating robots and uses communicated intents to predict potential future collisions. Robots move sequentially through the contention zone, which avoids collisions as long the robot that decided to yield does not create an impassable obstruction.

[1]Nokia Bell Labs, France
[2]Nokia Bell Labs, USA

II. Related Work

In industrial settings, commercial robot manufacturers [1], [2] expose an interface that can be used to define the constraints and interim goals (waypoints) such that the likelihood of contention is reduced. These constraints include forbidden zones, one-way pathways, exclusive-access zones, preferred lines and directions, speed-limit zones, and the like. Although traffic rules can help mitigate and partially avoid conflicts among a fleet of homogeneous robots, for a heterogeneous fleet of robots, the set of rules offered may differ. Hence, without explicit coordination with other robots, traffic rules are not enough to prevent collisions.

On the academic research side, there exists a wide literature on decentralized collision detection and avoidance, both algorithmic [3]–[8], and learning-based solutions [9]–[13]. A common thread for all academic solutions is that they are white box solutions. They are part of the robot's navigation stack, require access to sensor streams (*e.g.* LIDAR scans and/or camera-feeds), output motion control commands in the form of linear and angular velocities, and may even depend on internals of other subsystems such as localization [14]. Commercially available robots typically do not expose their internals nor interfaces at such a low level[1].

This makes existing academic solutions a non viable option for coordinating the navigation among a fleet of black-box robots. When the interface over which the robot can be observed and controlled is limited, the solution is forced to be picked from a limited set. Specifically, the only solution that can be universally made to work is to 1) observe the robot intent over available interfaces, 2) statistically predict where the robot is going to go, and 3) decide who stops to yield if a collision is anticipated.

CODAK offers a solution that is specifically designed to avoid tampering with the navigation stack and is limited to the monitoring and tasking information typically exposed by commercial robots. Further, CODAK's design leaves full

[1]Some black-box robots (e.g., MiR200) expose a remote control interface. Applying white-box solution via such an interface essentially means completely overriding the robot navigation stack which may not be possible due to lack of access to robot sensor streams and other input data required by most white-box solutions to compute collision-free velocity commands.

autonomy and control to the robot's internal navigation stack, and only steps in when a collision is anticipated. CODAK is also distributed with no single point of failure and has been validated using physical robots in real-world scenarios. One system also designed for black-box robots was reported by Pecora, *et al.* [15], which assumes that the robots follow fixed trajectories without re-planning. In contrast, we make no such assumptions and only require the robots to publish their planned trajectory to other robots.

III. CODAK

In absence of coordination, that is treating other robots as plain obstacles, robots may end up in a deadlock, exhibit oscillatory behavior known as the reciprocal dance [16], or collide due to uncertainty and sensor limitations. Our objective is to achieve the necessary coordination relying only on restricted interfaces and assuming no knowledge of inner workings of the robot's stack. The common assumption we base the design on is that each robot is capable of safely performing single-agent navigation in an obstacle-laden environment, as long as the obstacles are fully and timely detectable. We also assume that robots can localize themselves and report the location, planned path, and instantaneous velocity over the external interface.

The coordination is achieved via an external software module, which we call the *multi-robot handler* (MRH). Each MRH instance interfaces with a single robot and communicates with other MRHs. To make such a design viable, it is

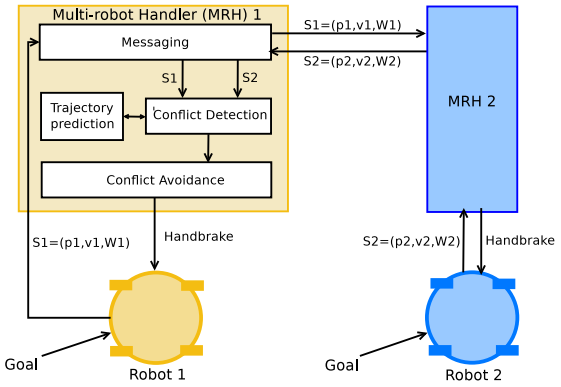


Fig. 1: Software architecture of CODAK.

useful to reflect on the interfaces that typical industrial robots expose. Any industrial robot, by construction, exposes an interface to monitor the robot's movements and an interface to issue and cancel a goal. By combining the two operations MRH can make the robot stop to yield by canceling the goal and re-issuing the goal after the yield time has completed. As an optimization, a dedicated *handbrake* interface that causes the robot to temporarily pause the motion may be available or easily conceivable.

Fig. 1 illustrates the main components of CODAK. The robots shown in the figure represent the hardware and software capable of performing single-agent navigation, given the externally supplied goal. As the robot progresses towards the goal, MRH monitors the progress, predicts potential collisions, and possibly pauses the robot using interfaces discussed above. Through these interfaces the MRH can 1) retrieve the robot's pose ($\mathbf{p}_i$), velocity vector ($\mathbf{v}_i$), and pursued plan ($\mathbf{W}_i$), and 2) engage or release the handbrake. The pursued plan is a set of waypoints on the path between the robot's present position and its goal. The pose, velocity, and pursued plan together form the robot's *intent*, S_i. In our notation we use index i to denote ith robot in the fleet.

The *messaging* module tracks the progress on the path and broadcast its intent and position. It also receives intents from other robots. Using this information, the *trajectory prediction* module calculates the distance to other robots. For robots in range (a subset of robots whose distance is lower than a predefined threshold), the trajectory prediction module estimates their future positions (Section III-A). The *conflict detection* module uses the predicted trajectory to pair-wise evaluate the collision probability (Section III-B). If a likely collision is detected, the *collision avoidance* module safely enforces a concurrency control scheme (Section III-C).

A. RNN-based Trajectory Prediction

The goal of the trajectory predictor is to estimate the position probability distribution over time $t \geq t_0$, given the ith robot's intent S_i at time t_0. This distribution is then used to determine whether the robot is on a collision course with another robot. Robot motion is subject to many stochastic processes, which are hard to capture in a closed-form model. Hence, we resort to training a recurrent neural network (RNN) model from observed data.

Let $\mathbf{W}_i = (\mathbf{w}_i(1), \cdots, \mathbf{w}_i(|\mathbf{W}_i|))$ be the pursued plan for ith robot, consisting of a sequence of waypoints $\mathbf{w}_i(k)$. For kth waypoint in the sequence, denoted as $\mathbf{w}_i(k)$, we define the corresponding kinetic state $\mathbf{y}_i(k) = (x, y, \theta, v, \omega, t)$ as the robot's state of motion when it is closest to that waypoint, where t is the time associated with the state, (x, y, θ) is the robot pose, and (v, ω) is the robot twist. We encode positions and timestamps as displacements from the first waypoint:

$$\mathbf{w}'_i(k) = \begin{cases} (0,0), & \text{if} \quad k = 0, \\ \mathbf{w}_i(k) - \mathbf{w}_i(k-1), & \text{if} \quad k > 0. \end{cases} \tag{1}$$

Similarly, we define the state displacement vector as $\mathbf{y}'_i(k) = (dx, dy, \theta, v, \omega, dt)$, where (dx, dy) is the relative position between (x, y) and waypoint $\mathbf{w}_i(k)$, and dt is the difference between the current and the previous state time.

The input to the RNN are the initial state $\mathbf{y}'_i(0)$ and the sequence of waypoints $\mathbf{w}'_i(0), \cdots, \mathbf{w}'_i(|\mathbf{W}_i|)$, all encoded as displacements. We model each state $\mathbf{y}'_i(k)$ as a multivariate normal distribution with mean $\boldsymbol{\mu}_i(k)$ and covariance $\boldsymbol{\Sigma}_i(k)$. The output of the RNN is a sequence of predicted means and covariances, one for each waypoint, denoted by $\tilde{\boldsymbol{\mu}}_i(k)$ and $\tilde{\boldsymbol{\Sigma}}_i(k)$. This follows the framework of mean-variance estimation first introduced in [17].

The structure of our neural network is illustrated in Fig. 2. It contains three modules: the encoder, mean decoder and covariance decoder. All modules are composed of multiple layers of Elman RNN cells with ReLU activation functions.

We apply layer normalization [18] to avoid exploding or vanishing gradients during training due to accumulation in the recurrent units, as conventionally done for RNNs. The encoder has 7 hidden layers of sizes 8, 16, 32, 64, 32, 16, and 8. The mean decoder and covariance decoder each have 4 hidden layers of sizes 16, 16, 32, and 32, followed by the final linear layer.

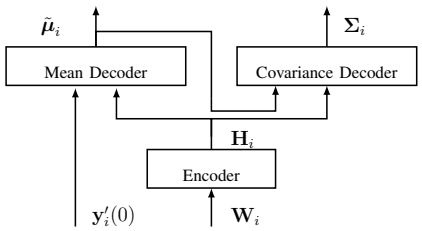
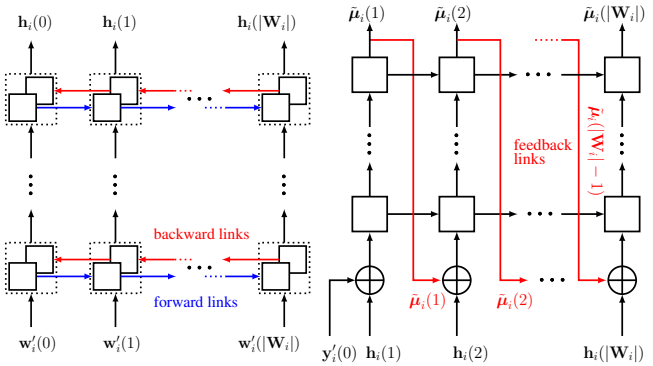


Fig. 2: Structure of the Trajectory Prediction Model

The encoder creates an internal representation $\mathbf{H}_i = \mathbf{h}_i(0), \cdots, \mathbf{h}_i(|\mathbf{W}_i|)$ of the pursued plan. At any iteration k, both past and future waypoints are necessary to predict the robot's state $\mathbf{y}'_i(k)$. Therefore, we design a bidirectional encoder in which successive hidden RNN cells are connected in both forward and backward directions (see Fig. 3a).



(a) Encoder (b) Mean Decoder

Fig. 3: Encoder and mean decoder modules. Each box represent an RNN cell with ReLU activation

The mean decoder is illustrated in Fig. 3b. It takes a pair $(\mathbf{y}'_i(0), \mathbf{h}_i(1))$ as input and outputs the predicted mean state $\tilde{\boldsymbol{\mu}}_i(1)$. This information is then fed back to the next iteration. At each iteration $k > 1$, the mean decoder processes $(\tilde{\boldsymbol{\mu}}_i(k-1), \mathbf{h}_i(k))$ and returns $\tilde{\boldsymbol{\mu}}_i(k)$. Due to this feedback link, the mean decoder is unidirectional, i.e., the outputs are computed sequentially for increasing k.

The covariance decoder uses the outputs of the encoder and the mean decoder to predict $\tilde{\boldsymbol{\Sigma}}_i(k)$, for all k. During our experiments, we observe that a simple unidirectional RNN with no feedback link achieves a good trade-off between accuracy and complexity. This is similar to the encoder's structure as shown in Fig. 3a but with unidirectional links.

For each robot known to the system we train the robot-type-specific model. To collect the training data, we run a single robot over 2000 randomly generated trajectories across three different maps: an empty map, a map with corridors (only 90-degree turns), and a maze-like map with obstacles of various shapes. Since the robot can re-plan, we collect the current plan and all future states on 2-second intervals, until the robot reaches the goal. One pair of pursued plan and sequence of states $(\mathbf{W}_i, \mathbf{Y}_i)$ is called a sample. We only keep samples with a plan of at least 4 meters.

90% of collected data are used to train the RNN model, 5% are used for validation and 5% for testing. Following [19], we perform two separate training phases. In the first phase, we train mean prediction using 75% of the training dataset with the mean square error loss. In the second phase, we freeze the encoder and mean decoder, and train the covariance decoder using the remaining 25% training samples, considering the Gaussian negative log likelihood loss. Both phases use the ADAM optimizer with a batch size of 128 and 100 epochs. Further, we apply a random rotation to the plans and states in each batch. This helps regularizing the model as it learns to predict trajectories regardless of their orientations. Once trained, the RNN model parameters are loaded into CODAK and they are valid for all robots that have the same mechanical properties.

B. Probabilistic Collision Detection

The RNN outputs the mean $\boldsymbol{\mu}_i(k)$ and covariance $\boldsymbol{\Sigma}_i(k)$ of predicted state associated with each waypoint $\mathbf{w}_i(k)$, where k is a common index in all three sequences. Recall that k also indexes into the timestamp t associated with the state. Of interest for collision detection is the mean and covariance of the robot position for an arbitrary timestamp t, which we denote as $\boldsymbol{\mu}_i^{\text{xy}}(t) = (x(t), y(t))$ and $\boldsymbol{\Sigma}_i^{\text{xy}}(t)$, and calculate by linearly interpolating the discrete-time states, followed by extracting the components of interest.

When the ith robot receives a new intent, $\mathbf{S}_j, j \neq i$, associated with timestamp, it sweeps the time parameter (using above-described interpolation) over the prediction of its own position $\boldsymbol{\mu}_i^{\text{xy}}(t), \boldsymbol{\Sigma}_i^{\text{xy}}(t)$ and the prediction of the positions of other robots $\boldsymbol{\mu}_j^{\text{xy}}(t), \boldsymbol{\Sigma}_j^{\text{xy}}(t), j \neq i$, and calculates the pair-wise probability of ith robot being at the same location at the same time as any jth robot as follows:

$$
\begin{aligned}
p_{ij}(t) &= \frac{1}{\eta_{ij}(t)} \int_{\mathcal{R}^2} \exp\Big(-\frac{1}{2}(\xi_i(t) + \xi_j(t))\Big) d\mathbf{x} \\
\xi_i(t) &= (\mathbf{x} - \boldsymbol{\mu}_i^{\text{xy}}(t))^{\text{T}} \boldsymbol{\Sigma}_i^{\text{xy}}(t) (\mathbf{x} - \boldsymbol{\mu}_i^{\text{xy}}(t)) \qquad (2) \\
\eta_{ij}(t) &= \frac{1}{(2\pi)^2 \boldsymbol{\Sigma}_i^{\text{xy}}(t) \boldsymbol{\Sigma}_j^{\text{xy}}(t)}
\end{aligned}
$$

For a point-mass robot, (2) has the closed form solution [20] and can be calculated efficiently in log-space. To account for the finite footprint, we (heuristically) adjust the covariance matrix by inflating the width and height of the 99.7% confidence ellipse by the robot diameter. We recompute the covariance matrix from the inflated ellipse and apply it in (2). The net effect of this adjustment is a median covariance ellipse area of at most 2x the robot footprint while achieving a 99.4% prediction accuracy (see Fig. 4).

Described heuristics is a tradeoff that allows us to still use the closed-form expression for (2), while reasonably approximating collision probability for a finite-footprint robot.

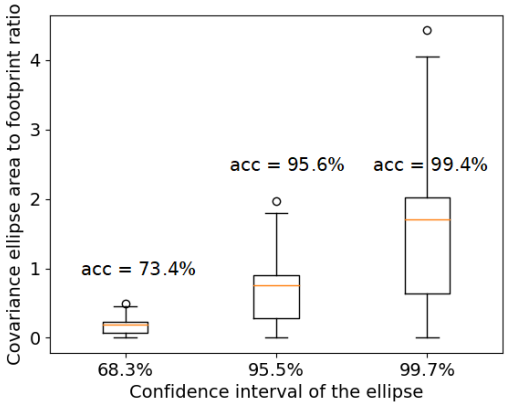


Fig. 4: RNN covariance ellipse area vs. confidence interval. Prediction accuracy is shown on top of each boxplot

If for any timestamp t, the calculated probability is greater than a set threshold, c_{th}, we declare that the robots are on a collision course.

C. Collision-Avoidance Protocol

Collision avoidance is a form of concurrent access control to a contention region in space-time. The contention can be resolved by 1) adjusting the velocity and trajectory and thereby proactively eliminating the contention region or by 2) stopping to yield and thereby blocking until the contention disappears. Fully customized motion planners [14], [21], [22] are amenable to (and indeed implement) the first mechanism, whereas the black-box system like CODAK is restricted to the latter mechanism. In CODAK each robot has an assigned priority (the specific priority assignment scheme is left for future work) that MRHs communicate to each other, which is used to decide which robot must stop to yield when the anticipated collision is detected.

The time to yield is a tradeoff between travel time makespan and the ability to prevent the contention. The time to achieve the collective goal will be the shortest if the robot postpones engaging the handbrake as long as physically possible. The lower bound of the distance from the collision point at which the robot must stop to yield is determined by the distance threshold d_{th}, defined as

$$d_{\text{th}} \geq (r_i + r_j) + s, \tag{3}$$

where s is the safety margin and r_i and r_j are the radii of the respective robot footprints. The safety margin accounts for localization uncertainty, finite stopping time (inertia), and collision-detection uncertainty (see Section III-B). The safety margin can guarantee collision-free motion if all random processes involved have a finite-tail distribution [23]. For infinite-tail distributions, it is only possible to guarantee the upper bound for collision probability, which decreases as s increases.

In our collision-avoidance protocol, s is a system parameter and the radii are mechanical parameters known by each robot and advertised to other robots. The MRH of the lower-priority robot continuously monitors its distance to the closest collision point that was detected along its trajectory. When the distance drops below d_{th}, lower-priority MRH engages the handbrake and releases it once the higher-priority robot moves away from the collision region[2].

This resolution protocol is iterated across all robots in the vicinity; it thereby successfully resolves contentions involving multiple robots. In Section IV, we show experimental results involving four robots. One limitation of our work is that we assume that locations at which robots stop to yield have enough free space to not create impassable obstructions. This assumption holds if CODAK is configured with predefined safe-to-stop zones. For future work, we aim to extend CODAK to identify such zones dynamically, thereby enabling conflict-resolution without static provisioning.

IV. Experimental Evaluation

We divide the evaluation into two parts: first we evaluate the accuracy of trajectory prediction and next we evaluate the system as a whole.

A. Trajectory Prediction Error

Because collision detection depends on predicting the robot position, which in turn depends on trajectory prediction, it is important to experimentally verify that the model works. We ran a simulation experiment of a robot in Gazebo [24] with a hand-crafted high-fidelity model that captures full mechanical effects of the actual physical robot and the environment it operates in. Evaluating the trajectory prediction in simulation allows us to compare against the ground truth, which is difficult or impossible to obtain on live robots.

We compare the RNN model described in Section III-A to a model that extrapolates the robot position using the mean velocity over the past second (AvgVel). Fig. 5 shows the Euclidean distance between the predicted robot position and the ground truth obtained from the simulator. The lines represent the mean error as a function of elapsed time and the shaded areas represent the standard deviation over multiple tests collected in a maze-like environment.

As expected, prediction error grows as the prediction horizon extends further into the future. RNN consistently outperforms the AvgVel model both in the mean and standard deviation, so it is more accurate and more confident.

The primary source of error in AvgVel comes from robots slowing down when they make turns and speeding up on a straight-line path. This causes the AvgVel to underestimate or overestimate the robot position for several timesteps. Further, robots may deviate from the planned paths, because the local planner can decide to cut corners, or a dynamic obstacle may cause the robot to make evasive actions.

Corner-cutting and other deviations from the global plan explain the finite error for AvgVel model at time $t = 0s$. Because the start of the prediction window is typically a point in time after the robot has already been moving, it is possible

[2]If a native handbrake interface is available, MRH can continuously reissue the goal while retaining the handbrake to invoke safe replanning. If an alternative conflict-free path is found, the handbrake can be released sooner.

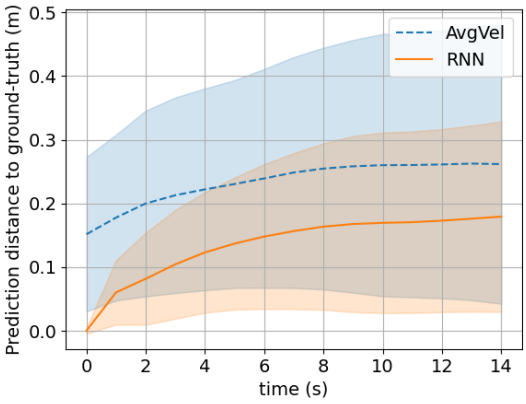


Fig. 5: Prediction error over time of RNN and AvgVel

and likely that the AvgVel will produce an error even for $t = 0$. In contrast to AvgVel, RNN takes the plan waypoints into account and learns subtle interactions between the global and local planner, which results in better prediction.

B. Experimental Setup

We evaluate CODAK mostly in simulation and repeat a few test cases on live robots. Simulation allows us to collect the data more efficiently as tests are fully automated and we do not need to spend time placing robots at consistent start position, whereas repeating a limited number of tests on live robots validates the simulation method. In total, we repeat each test case 30 times in simulation, and 5 times with live robots. Throughout all our tests, we set the prediction horizon to 5s, the distance threshold d_{th} to 2m, and the collision threshold c_{th} to 0.0001. The robot radius is 0.35m.

For live robot tests, we use Jackal robots from Clearpath Robotics [25], equipped with Velodyne VLP-16 LIDAR [26], which is used for localization and obstacle detection. The localization algorithm is an in-house developed Kalman filter [27], which uses the robot's inertial odometry (IMU and wheel-encoders) for prediction and compares the LIDAR scan with the environment map using the PL-ICP algorithm [28] in the measurement stage. The navigation stack is the publicly available ROS navigation stack [29]. For simulation tests, we use Gazebo [24] with dynamic robot model that has been tuned to match the behavior observed on live robots. Identical software for sensor data acquisition and motor actuation runs on simulated and live robots. Although ROS is an open system, for CODAK evaluation, we restrict ourselves to using only the interfaces defined in Section III and thereby treat the navigation stack as a black box.

We also ported the COCALU [14] planner code [30] to our robots, so that we can compare it with our solution in identical environment and the same inter-robot communication protocol. COCALU is a collision-avoidance planner based on the velocity obstacle (VO) notion [21]. A VO is the set of all velocities that lead to a collision with an obstacle. In a VO-based solution, each robot computes a VO for each of the other robots, and adapts its velocity over time to remain outside of all VOs. We deactivated COCALU's footprint inflation feature thereby reducing it to NH-ORCA [22] (a popular benchmark in collision avoidance literature).

C. System Evaluation

We evaluate the system in an indoor environment using the four representative test scenarios shown in Fig. 6, most of which have been widely used in previously published work [8]–[10], [12], [13]. Circles with markers in the center indicate start positions and matching markers the goals.

For each scenario, we evaluate three systems. Baseline configuration is the "stock" ROS navigation stack [29] which comes with no explicit collision avoidance. Robots are left to their sensors (typically LIDAR) to spot and treat each other as any other obstacle. Blind spots or late observation of full robot footprint often cause collision in this case. The main purpose of the baseline case is to establish the need for CODAK coordination policy. CODAK non-invasively extends the single-robot planner with multi-robot coordination capabilities, but retains their autonomy. NH-ORCA [22] is a representative planner specifically designed for multi-robot coordination. We consider the following key performance metrics:

- Success rate: Success rate indicates the ratio of successful experiments to the total number of experiments performed. For each scenario-system pair, we repeat the experiment multiple times, and record whether each run is successful. An experiment is deemed successful if all robots reach their goals. In unsuccessful experiments, at least one robot failed to reach the goal, either due to deadlocks or collisions.
- Makespan: We also record the time it takes for each robot to go from start to goal, and compute the makespan as the time elapsed between when the goals are issued until the last robot reaches its goal. We report the average makespan with 95% confidence interval (CI) for each scenario-system pairs.
- Total Distance: Finally, we record the trajectory followed by each robot, and report the average sum of distances traveled in meters with 95% CI across all runs for a scenario-system pair.

The results in Table I demonstrate the benefit of explicit coordination. Although ROS navigation stack is designed to enable robots to detect and react to obstacles, treating other robots as obstacles is clearly not enough, as evident by the low success rate of ROS-Nav. The small number of collision cases (2.5% of all runs) reported with CODAK occurred for reasons unrelated to collision prediction and avoidance. The low-priority robot successfully anticipated the collision and stopped, effectively turning itself into a static obstacle. The collision that followed in these rare cases could have easily happened in a single-agent scenario. This problem can be resolved with tuning the planner parameters; however we avoid touching the navigation stack since CODAK is designed for black-box robots.

The two scenarios that we ran with live robots have showed almost identical results compared to simulations,

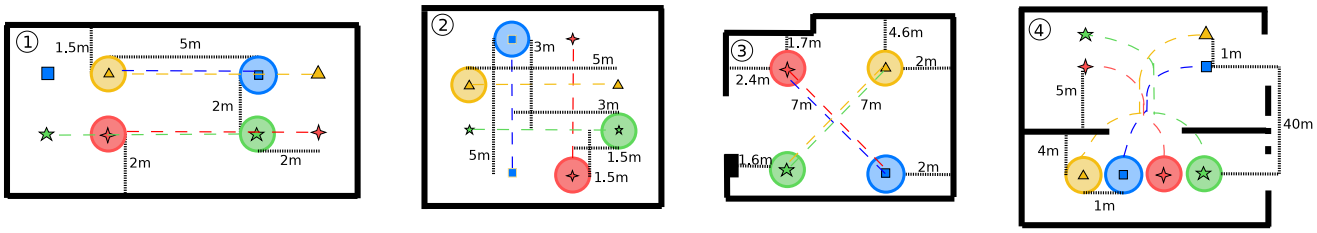
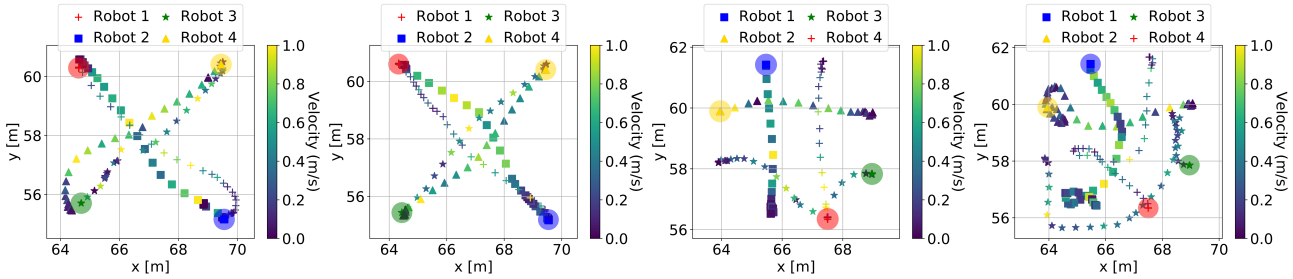


Fig. 6: Test Scenarios: (1) Head-to-Head (HH), (2) Intersection (I), (3) Antipodal Circle (AC), (4) Narrow Pathway (NP).

Map	CODAK			ROS-Nav			NH-ORCA		
	Success Rate	Makespan	Total Distance	Success Rate	Makespan	Total Distance	Success Rate	Makespan	Total Distance
Simulation Runs									
AC	30/30	37.87±0.77	34.55±0.47	1/30	31	47	30/30	**27.13±2.22**	35.23±0.86
I	30/30	**27.47±0.64**	24.01±0.27	0/30	-	-	28/30	29.5±2.15	33.18±1.72
NP	29/30	46.38±2	43.65±0.99	2/30	50.5	60.53	29/30	**40.0±1.47**	45.97±1.03
HH	28/30	**25.39±0.73**	30.84±0.23	0/30	-	-	30/30	28.13±1.74	35.4±0.7
Live Runs									
NP	5/5	46.8±4.13	45.04±1.36	0/5	-	-	5/5	**42.8±1.16**	45.03±0.45
HH	5/5	**25.8±2.03**	29.74±0.38	1/5	19.0	29.68	5/5	27.0±1.57	33.84±0.62

TABLE I: CODAK vs. ROS-Nav vs. NH-ORCA



(a) Antipodal Circle CODAK (b) Antipodal Circle NH-ORCA (c) Intersection CODAK (d) Intersection NH-ORCA

Fig. 7: Robot trajectories achieved for different test scenarios.

which validates our models. For antipodal-circle (AC) and narrow pathway (NP) scenarios, CODAK incurs a penalty of 11.6s and 5.15s, respectively. Inspecting the trajectories, we observed that robots with CODAK and NH-ORCA follow similar paths (Fig. 7a and Fig. 7b), so the time penalty shows the advantage of speed adjustment over stop-and-yield policy.

For intersection (I) and head-to-head (HH) scenarios, we observe shorter makespan for CODAK, which we initially found surprising. Inspecting the trajectories shows that stop-and-yield strategy actually led to smoother trajectories (Fig. 7c and Fig. 7d). These results demonstrate that CODAK is not limited to black-box robots, and can indeed be used in white-box robots as part of a combined planner that can contextualize when stop-and-yield is a better response than exaggerated velocity controls.

V. Conclusions

CODAK attacks the problem of multi-robot coordination when controlling the motion is restricted to tasking interface with no direct control of robot velocity. This has practical value for closed-system robots whose localization and navigation algorithms are kept proprietary and also allows interoperability among robots from multiple vendors (heterogeneous fleet). Planners are simply required to meet reasonable common expectations, namely, to produce paths that get the robot from one point to another while respecting the cost imposed by obstacles and to expose monitoring and tasking interfaces. Within these expectations, CODAK is compatible with any black box robot.

Within the stated restriction CODAK is forced to implement the stop-and-yield collision avoidance policy, whereas planners with explicit coordination can use more sophisticated evasive action, such as mutual velocity adjustment or path negotiation. We expected CODAK to incur a performance penalty compared to a velocity-adjusting planner. In most cases we saw a minor penalty when tested under otherwise identical conditions.

A somewhat surprising result was that in certain scenarios CODAK performed better in terms of covered distance and makespan. Our explanation of this observation is that certain scenarios are simply amenable to stop-and-yield policy. In these scenarios we saw a velocity-adjusting planner spin the robots in place or even move them backwards, inflicting the penalty on itself. We have shown that coordination among multiple single-agent, closed-system, robots is feasible, without significant performance penalty.

REFERENCES

[1] OMRON, "Fleet operations workspace core user's manual." Available at https://assets.omron.eu/downloads/manual/en/v11/i635_fleet_operations_workspace_core_-_user_guide_users_manual_en.pdf, version 11.0.

[2] M. I. Robots, "Mir fleet interface reference guide." Available at https://www.mobile-industrial-robots.com/media/13757/mir-fleet-reference-guide-15_en.pdf, version 1.5.

[3] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. and Autom.*, pp. 1928–1935, IEEE, 2008.

[4] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Optimal reciprocal collision avoidance for multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. and Autom.*, 2010.

[5] K. Guo, D. Wang, T. Fan, and J. Pan, "Vr-orca: Variable responsibility optimal reciprocal collision avoidance," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4520–4527, 2021.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Dist. Auton. Robot. Sys.*, pp. 203–216, Springer, 2013.

[7] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. and Sys. (IROS)*, pp. 5917–5922, IEEE, 2009.

[8] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robot. Research*, pp. 3–19, Springer, 2011.

[9] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. and Autom.*, pp. 6252–6259, IEEE, 2018.

[10] Q. Tan, T. Fan, J. Pan, and D. Manocha, "Deepmnavigate: deep reinforced multi-robot navigation unifying local & global collision avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. and Sys. (IROS)*, pp. 6952–6959, IEEE, 2020.

[11] R. Han, S. Chen, S. Wang, Z. Zhang, R. Gao, Q. Hao, and J. Pan, "Reinforcement learned distributed multi-robot navigation with reciprocal velocity obstacle shaped rewards," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 5896–5903, 2022.

[12] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. and Sys. (IROS)*, pp. 11785–11792, IEEE, 2020.

[13] J. Blumenkamp, S. Morad, J. Gielis, Q. Li, and A. Prorok, "A framework for real-world multi-robot systems running decentralized gnn-based policies," *arXiv preprint arXiv:2111.01777*, 2021.

[14] D. Claes and K. Tuyls, "Multi robot collision avoidance in a shared workspace," *Autonomous Robots*, vol. 42, no. 8, pp. 1749–1770, 2018.

[15] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, pp. 485–493, 2018.

[16] F. Feurtey, "Simulating the collision avoidance behavior of pedestrians," *Master's thesis, University of Tokyo, Department of Electronic Engineering*, 2000.

[17] D. A. Nix and A. S. Weigend, "Estimating the mean and variance of the target probability distribution," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 1, pp. 55–60, IEEE, 1994.

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[19] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, "Comprehensive review of neural network-based prediction intervals and new advances," *IEEE Trans. Neural Netw.*, vol. 22, no. 9, pp. 1341–1356, 2011.

[20] P. Bromiley, "Products and convolutions of gaussian probability density functions," *Tina-Vision Memo*, vol. 3, no. 4, p. 1, 2003.

[21] 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.

[22] D. Hennes, D. Claes, W. Meeussen, and K. Tuyls, "Multi-robot collision avoidance with localization uncertainty.," in *Auton. Agents and Multi-Agent Sys.*, pp. 147–154, Springer, 2012.

[23] K. Shah, G. Angeris, and M. Schwager, "Reciprocal multi-robot collision avoidance with asymmetric state uncertainty," *arXiv preprint arXiv:2107.10956*, 2021.

[24] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. and Sys. (IROS)*, pp. 2149–2154, Sep. 2004.

[25] "Jackal unmanned ground vehicle." ClearPath Robotics, Product Datasheet. available at https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/.

[26] "Vlp-16 user manual." Vendor Documentation, 2019. Available at https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf.

[27] "ROS lsm localization." Web Page (ROS Index), 2022. available at https://index.ros.org/p/lsm_localization/.

[28] A. Censi, "An icp variant using a point-to-line metric," in *Proc. IEEE Int. Conf. Robot. and Autom.*, pp. 19–25, May 2008.

[29] "ROS navigation package." Web Page (ROS documentation), 2020. available at http://wiki.ros.org/navigation.

[30] "Multi-robot collision avoidance." Web Page (ROS documentation), 2015. available at http://wiki.ros.org/multi_robot_collision_avoidance.