Obstacle Avoidance with Kinetic Energy Buffer

V. Pitkänen, T. Pennanen, A. Tikanmäki, J. Röning

Abstract— **This paper presents Kinetic Energy Difference (KED) as a metric for collision proximity. The calculation of KED for differentially driven robots is explained, along with an example obstacle avoidance algorithm that utilizes it. This example algorithm is computationally efficient and simulations show that it is capable of guiding robots with slow dynamics through narrow corridors.**

I. Introduction

Choosing a suitable obstacle avoidance method for a mobile robot in an unknown environment is not a straightforward task [1], and the choice is typically a compromise between different attributes such as speed and safety. The ideal method in each case depends on the structure of the environment, the dynamic capabilities and the shape of the robot, and the sensors used to observe the obstacles. The computational capacity of the robot is an additional constraint when selecting a feasible method. To avoid computational complexity, a typical approach is to make some approximations of the shape, size and kinematics of the robot, or to ignore these properties altogether.

Obstacle avoidance is intertwined with the problem of path planning. There are no strict definitions, but it is generally considered that global planners work off-line to calculate some long-term path based on a priori data of the robot's surroundings, while the goal of the local planners is to combine these path and sensor readings to generate moment-to-moment control signals to the actuators. Obstacle avoidance is a critical part of a local planner and needs to maintain a good balance between collision prevention and local goal-seeking navigation.

Perhaps the most well-known and widely used family of obstacle avoidance methods is that based on the Artificial Potential Field (APF) method [3]. In their most basic form, APF methods generate virtual forces that repel the robot away from obstacles and attract it towards a goal. Variations of APF include approaches where the potential function is modified to account for the robot's velocity as well as position [4] or to allow the treatment of moving targets and obstacles [5]. APF methods are generally simple and very fast, but suffer from the local minima problem where the repelling and attracting virtual forces are equal, and the robot stops before reaching its goal. Furthermore, APF methods often do not consider the exact shape of the robots, nor take into consideration that differentially driven robots are actually not holonomic.

V. Pitkänen is with the Biomimetics and Intelligent Systems Group, University of Oulu, Finland. (ville.pitkanen@oulu.fi)

T. Pennanen is with the Biomimetics and Intelligent Systems Group, University of Oulu, Finland. (tuulia.pennanen@oulu.fi)

A. Tikanmäki is with the Biomimetics and Intelligent Systems Group, University of Oulu, Finland. (antti.tikanmaki@oulu.fi)

J. Röning is with the Biomimetics and Intelligent Systems Group, University of Oulu, Finland. (juha.roning@oulu.fi)

Another well-known group of obstacle avoidance methods is based on the Dynamic Window Approach (DWA) [6]. Such methods usually not only take into account the exact shape of the robot, but also consider the velocity and acceleration limits. In these methods the steering commands are calculated in the robot's velocity space by first generating feasible circular arc trajectories, and then picking the best one according to some optimization function. By using circular arcs, the issue of the non-holonomy of differentially steered robots is explicitly addressed. One significant variation of DWA is the Velocity Obstacles method [7], which includes observations of obstacle motions in the evaluation of safe trajectories. The downside of DWA type methods is that as they are based on finding minima of cost-functions, the available velocity commands are discretized, with higher resolutions requiring increasing processing power.

Some methods with long histories in plant control have been modified to become applicable to the obstacle avoidance problem. One such example is the Model Predictive Control (MPC) approach [2]. MPC methods can take into account exact shapes and dynamical capabilities, but are often complex and computationally very expensive because before each motion command the method basically calculates multiple simulations forward in time.

Other methods like Arc Reachable Manifolds (ARM) [8], and Parameterized Trajectory Generators (PTG) [13][14] abstract away the robot's physical properties so that controllers can consider them holonomic points on a plane, and thus making a wide variety of controllers, such as those based on APFs, applicable. These methods, while generally advanced and not computationally expensive, however do not consider the limited motor torques and possible environmental disturbances explicitly.

In this paper, we consider differentially driven robots with slow dynamics, and as the main contribution present a new metric for collision proximity, Kinetic Energy Difference (KED), which can be used in their obstacle avoidance. Of the previously mentioned methods, the metric and its use most closely resemble [8], [13] and [14] in the sense that the metric operates in abstract spaces to which the robot and obstacles have been translated. The new metric accounts for the shape, torque limits and other dynamics of the robot. This paper also presents one example motion controller that utilizes the new metric. The example controller is computationally inexpensive, works in unknown environments, and anticipates environmental disturbances (such as rough terrain) that can subject the robot to unexpected but limited energy injections that change the robot's translational and rotational kinetic energy. The paper is organized as follows: Section II introduces the metric, section III gives an example controller that utilizes it, section IV shows simulation results, section V shows performance with an actual robot, and section VI concludes the paper.

II. KINETIC ENERGY DIFFERENCE

This section explains the concept and calculation of KED.

A. Models

Figure 1 shows the robot and environmental models. The robot body is rigid and non-deformable, and has a coordinate frame {B} attached to it between the two powered wheels w_1 and w_2. Unless otherwise noted all values are expressed in {B} and are relative to it. The wheels are located at $(0, y_1)$ and $(0, y_2)$, and can produce maximum nominal forces f_1 and f_2 with the ground. There are M (green) points b_i rigidly attached to {B}. These "bumper" points are the (usually slightly enlarged) approximation of the robot's body. Note that although the robot has a box-like shape in Fig. 1, b_i could of course be used to approximate any shape. The robot's mass m is centered at {B} and the robot has a rotational inertia of I. At any given moment the robot has a translational velocity of v and rotational velocity of ω. The robot is given "soft" maximum velocity and acceleration limits v_{max}, ω_{max}, a_{max} and Φ_{max} .The environment is planar, has an inertial coordinate frame {G} attached to it, and has a varying total of N (red) known obstacle points o_j.

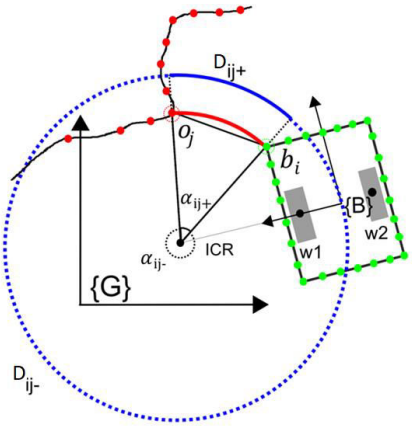

Figure 1. World and robot models.

B. Maximum Recoverable Initial Velocities

For each pair of b_i and o_j, there is a circular track with radius R_{ij} that would cause a collision between the two points. This collision can happen if the robot either goes forward a distance of D_{ij+} or backward a distance of D_{ij-}. Dependent on the robot's dynamical capabilities, there are two maximum initial velocities, the positive forward v_{ij+} and the negative backward v_{ij-}, from which the robot can decelerate to full stop before collision while maintaining the circular track of radius R_{ij}. The rest of this section explains how to calculate v_{ij-} and v_{ij+} for each b_i and o_j pair.

R_{ij} (i.e., the y coordinate of the Instantaneous Center of Rotation (ICR) in {B}) is calculated as

$$R_{ij} = \frac{x_{oj}^2 - x_{bi}^2 + y_{oj}^2 - y_{bi}^2}{2(y_{oj} - y_{bi})}.$$

With differential steering, sideways motion is prohibited, so the associated Lie algebra (see e.g. [10]) and its exponential mapping have the matrix forms

$$\varepsilon_{ij} = \begin{bmatrix} 0 & -\alpha_{ij} & \alpha_{ij} R_{ij} \\ \alpha_{ij} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \in se(2),$$

$$\exp(\varepsilon_{ij}) = T_{ij} = \begin{bmatrix} \mathrm{C_{ij}} & -\mathrm{S_{ij}} & R_{ij} S_{ij} \\ \mathrm{S_{ij}} & \mathrm{C_{ij}} & R_{ij}(1 - C_{ij}) \\ 0 & 0 & 1 \end{bmatrix} \in SE(2),$$

where $\mathrm{C_{ij}} = \cos(\alpha_{ij})$ and $\mathrm{S_{ij}} = \sin(\alpha_{ij})$. If the robot rotates an angle α_{ij} about the ICR located at $(0, R_{ij})$, then the body point (x_{bi}, y_{bi}) will move to

$$\begin{bmatrix} x'_{bi} \\ y'_{bi} \\ 1 \end{bmatrix} = T_{ij} \begin{bmatrix} x_{bi} \\ y_{bi} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{bi}\mathrm{C_{ij}} - y_{bi}\mathrm{S_{ij}} + R_{ij}\mathrm{S_{ij}} \\ x_{bi}\mathrm{S_{ij}} + y_{bi}\mathrm{C_{ij}} + R_{ij}(1 - \mathrm{C_{ij}}) \\ 1 \end{bmatrix}$$

Therefore, for b_i and o_j to collide the following group equation must be true:

$$\begin{cases} x_{bi}\mathrm{C_{ij}} + (R_{ij} - y_{bi})\mathrm{S_{ij}} = x_{oj} \\ (-R_{ij} + y_{bi})\mathrm{C_{ij}} + x_{bi}\mathrm{S_{ij}} = y_{oj} - R_{ij} \end{cases}.$$

The above equals the following due to the rules regarding the linear combinations of sines and cosines:

$$\begin{cases} H_{ij} cos(\alpha_{ij} - \varphi_{ij1}) = x_{oj} \\ H_{ij} cos(\alpha_{ij} - \varphi_{ij2}) = y_{oj} - R_{ij} \end{cases},$$

$$H_{ij} = \sqrt{x_{bi}^2 + (R_{ij} - y_{bi})^2},$$

$$\varphi_{ij1} = arctan2(R_{ij} - y_{bi}, x_{bi}),$$

$$\varphi_{ij2} = arctan2(x_{bi}, -R_{ij} + y_{bi}).$$

Solving for α_{ij} results in

$$\begin{cases} \alpha_{ij} = \arccos\left(\frac{x_{oj}}{H_{ij}}\right) + \varphi_{ij1} = \lambda_{ij1} + \varphi_{ij1} \\ \alpha_{ij} = \arccos\left(\frac{y_{oj} - R_{ij}}{H_{ij}}\right) + \varphi_{ij2} = \lambda_{ij2} + \varphi_{ij2} \end{cases}$$

In most programming languages arccos returns a value in the range $[0, \pi]$. In that case, there are four relevant angle values for each pair of b_i and o_j as $\cos(\alpha_{ij}) = \cos(-\alpha_{ij})$;

$$\beta_{ij1+} = f_{[0,2\pi)}(\lambda_{ij1} + \varphi_{ij1}),$$

$$\beta_{ij1-} = f_{[0,2\pi)}(-\lambda_{ij1} + \varphi_{ij1}),$$

$$\beta_{ij2+} = f_{[0,2\pi)}(\lambda_{ij2} + \varphi_{ij2}),$$

$$\beta_{ij2-} = f_{[0,2\pi)}(-\lambda_{ij2} + \varphi_{ij2}),$$

where the function $f_{[0,2\pi)}$ simply sets the sum to the range $[0,2\pi)$. Points b_i and o_j will collide if the robot is rotated about the ICR at $(0, R_{ij})$ by

$$\alpha_{ij} = \begin{cases} \beta_{ij1-}, if\ (\beta_{ij1-} = \beta_{ij2+})\ or\ (\beta_{ij1-} = \beta_{ij2-}) \\ \beta_{ij1+}, else \end{cases},$$

so the "forward travel angle" as seen in Figure 1 is

$$\alpha_{ij+} = \begin{cases} a_{ij}, if\ R_{ij} > 0 \\ 2\pi - a_{ij}, else \end{cases}.$$

The forward and backward collision distances for the robot are

$$D_{ij+} = |R_{ij}\ \alpha_{ij+}|,$$
$$D_{ij-} = |R_{ij}\ (2\pi - \alpha_{ij+})| = |R_{ij}\ \alpha_{ij-}|.$$

A robot on a circular track has the following maximum path deceleration due to the limited forces the wheels can produce with the ground:

$$a_{f,ij} = -\min(|a_1|, |a_2|) \quad (1)$$

$$a_1 = \frac{f_1(-y_1+y_2)}{\frac{I}{R_{ij}}+my_2}, a_2 = \frac{f_2(y_1-y_2)}{\frac{I}{R_{ij}}+my_1}. \quad (2)$$

The robot's path deceleration is also limited by a_{max} and Φ_{max} so the final maximum deceleration is

$$a_{ij} = -\min(\ |a_{f,ij}|, a_{max}, \Phi_{max}|R_{ij}|). \quad (3)$$

Therefore, a robot on a circular collision track as described by R_{ij} and α_{ij+} can brake down to zero velocity on the "forward" arc of length D_{ij+}while maintaining R_{ij} if it has at maximum the initial velocity

$$v_{ij+} = \sqrt{-2D_{ij+}a_{ij}}\ , \omega_{ij+} = \frac{v_{ij+}}{R_{ij}}. \quad (4)$$

Similarly for the "backward" arc of length D_{ij-}

$$v_{ij-} = -\sqrt{-2D_{ij-}a_{ij}}\ , \omega_{ij-} = \frac{v_{ij-}}{R_{ij}}\ . \quad (5)$$

Note that in this paper the robot has a symmetrical mass distribution centered at {B}. If that is not the case, the equations (1-5) need to be modified. Examples how to calculate wheel forces for robots with an asymmetrical mass distribution are found for example in [9].

C. Kinetic Energy Space

Given m, I , v, and ω, the corresponding translational and rotational kinetic energy can be mapped to points in $\mathbb{R}^2$, from now on called the Kinetic Energy Space (KES). As shown in Fig. 2, if the robot has the velocity (v, ω), then its kinetic energy state is mapped to $E_{robot} = (E_v, E_\omega)$ where $E_v = sign(v)0.5mv^2$ is the coordinate on the horizontal axis and $E_\omega = sign(\omega)0.5I\omega^2$ the coordinate on the vertical axis. The recoverable initial velocities (v_{ij+}, ω_{ij+}) and (v_{ij-}, ω_{ij-}) get mapped to points $E_{ij+} = (E_{vij+}, E_{\omega ij+})$ and $E_{ij-} = (E_{vij-}, E_{\omega ij-})$ in a similar fashion. Note that the mapping takes into account the sign of the velocity component, e.g., if $v<0$, then the mapped point in KES is on the left horizontal half-plane, thus indicating the robot has a "negative" translational kinetic energy. For a unicycle type robot with a fixed mass and rotational inertia, each circular path with a fixed radius must have a fixed ratio between $E_\omega$and E_v, i.e., $E_\omega = KE_v$, thus each R_{ij} is represented in KES as a line through the origin with a slope of

$$K_{ij} = sign(R_{ij})\frac{I}{mR_{ij}^2}.$$

Therefore, all the points on the rays starting at E_{ij+} and E_{ij-} with the slope of K_{ij} are energy states which would result in a collision under the assumption that R_{ij} were maintained throughout the braking.

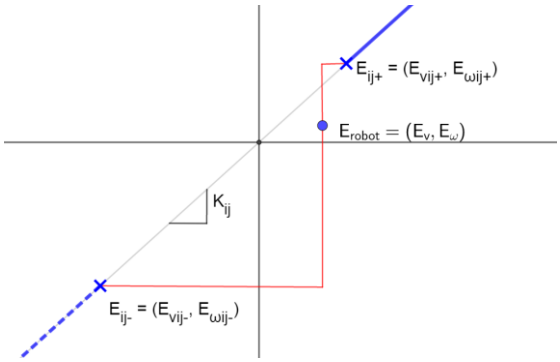

Figure 2. Kinetic Energy Space. For clarity, the mapping of only one b_i and o_j pair is shown but there are actully MN lines through the origin and 2MN +1 points (including the robot's energy state E_{robot}).

D. Kinetic Energy Difference

As E_{ij+} and E_{ij-} are the robot's energy states from which it can barely decelerate to stop before collision, they (and the rays starting from them) can be viewed as dangerous obstacle states that E_{robot} should avoid, analogous to how the robot's physical body should stay away from walls and other physical obstacles. We propose the *Kinetic Energy Difference* (KED) as a metric of collision proximity for differentially steered robots. KED is the Manhattan distance (L1 norm) between two points in KES, i.e., the total amount of energy that is needed to change the robot's energy state from one point to the other. Of course, instantaneous energy injection is physically impossible, but many phenomena, such as rolling over low obstacles, can cause significant energy injections in a short time period. Even when the robot is not reasonably expected to encounter such disturbances, KED remains a suitable metric as it incorporates rich information about obstacle proximity, shape, velocity, and dynamic limits.

III. Example Robot Motion Control With KED

This section gives one, conceptually as simple as possible, example on how KED can be utilized to control a robot with unicycle kinematics and limited dynamic capabilities. As will be shown later, the example controller is computationally inexpensive enough even for affordable hardware. The example consists of two main parts. The velocity limiter (section III.A), which calculates the maximum allowed "safe" twist for the robot, and steering (section III.B), which calculates the twist the robot should take to get closer to the waypoint while avoiding obstacles. The final output of the example controller is an acceleration twist (section III.C).

Each (b_i, o_j) pair produces two eventual collision states in KES, namely the points E_{ij+} and E_{ij-}. For the sake of readability, from here on the $+/-$ subscripts are replaced with a binary variable s (as in sign). E_{ijs} is now a generic collision state in KES, and KED_{ijs} the L1 distance between it and E_{robot}.

A. Velocity Limits

All KED_{ijs} and E_{ijs} are divided to sets according to which of the four quadrants of KES the corresponding E_{ijs} are in:

$$KED_{Q1} = \{KED_{ijs} | E_{vijs} > 0 \; and \; E_{\omega ijs} > 0\},$$
$$KED_{Q2} = \{KED_{ijs} | E_{vijs} > 0 \; and \; E_{\omega ijs} < 0\},$$
$$KED_{Q3} = \{KED_{ijs} | E_{vijs} < 0 \; and \; E_{\omega ijs} < 0\},$$
$$KED_{Q4} = \{KED_{ijs} | E_{vijs} < 0 \; and \; E_{\omega ijs} > 0\},$$
$$E_{Q1} = \{E_{ijs} | E_{vijs} > 0 \; and \; E_{\omega ijs} > 0\},$$
$$E_{Q2} = \{E_{ijs} | E_{vijs} > 0 \; and \; E_{\omega ijs} < 0\},$$
$$E_{Q3} = \{E_{ijs} | E_{vijs} < 0 \; and \; E_{\omega ijs} < 0\},$$
$$E_{Q4} = \{E_{ijs} | E_{vijs} < 0 \; and \; E_{\omega ijs} > 0\}.$$

KED_{min} and KED_{max} are tuning parameters which describe the energy injection the robot *must* and *should* always tolerate without crashing. For each control cycle the robot's allowed velocities are calculated with

$$v_{alwd,+} = \max(0, v_{max} \min(1, \frac{\min(M_1, M_2) - KED_{min}}{KED_{max} - KED_{min}}),$$
$$v_{alwd,-} = -\max(0, v_{max} \min(1, \frac{\min(M_3, M_4) - KED_{min}}{KED_{max} - KED_{min}}),$$
$$\omega_{alwd,+} = \max(0, \omega_{max} \min(1, \frac{\min(M_1, M_4) - KED_{min}}{KED_{max} - KED_{min}}),$$
$$\omega_{alwd,-} = -\max(0, \omega_{max} \min(1, \frac{\min(M_2, M_3) - KED_{min}}{KED_{max} - KED_{min}}),$$
$$M_n = \min(\min(KED_{Qn}), \min(E_{Qn})).$$

$v_{alwd,+}$, $v_{alwd,-}$, $\omega_{alwd,+}$ and $\omega_{alwd,-}$ effectively construct four walls around the origin of KES and E_{robot} is allowed to be inside the "box" they construct. Small KED_{ijs} and E_{ijs} therefore slow down the robot. Usually E_{ijs} have no effect but their inclusion is critical as without them the controller could ignore E_{ijs} that are close to the origin (thus implying close physical proximity) if E_{robot} is far away from it. For example, a sudden appearance of an obstacle in front of a robot with a high velocity could cause such a scenario.

B. Steering

The example controller guides the robot through narrow passages using a (possibly inaccurate) continuous or discrete path, which is given by some global planner. A waypoint W moves forward on the path if the distance between it and {B} is less than $k_{pathDist}$. However, W is informally speaking just a weak recommendation where to go, and mostly used just to divide obstacle points to those that should be avoided by turning left, and those that should be avoided by turning right.

1) Left Obstacles and Right Obstacles

A sensor akin to a 2D LIDAR is assumed to give the distances and direction angles of obstacle points. Then a "virtual corridor" of width $k_{minDist}$ is generated between {B} and W. $k_{minDist}$ is the minimum opening size the robot is allowed to go through and the angle to W is β_W. If this corridor does not contain any obstacle points, then the line that divides o_j to "left obstacles" and "right obstacles" is in the direction $\beta_D = \beta_W$. If the virtual corridor has any o_j, then the closest one along the virtual corridor is picked. This closest obstacle point o_c is then used as a "seed" from which the obstructing continuous obstacle co is constructed. This construction is done in three steps (see Figure 3 for illustration). The first step is to use o_c as the starting point to grow a graph-like structure whose nodes are obstacle points (red points), and two of these nodes are connected (red lines) if they are closer than $k_{minDist}$ to each other. The second step is to go through each node in the structure and calculate the circular sector that contains it and all the nodes it is connected to (see Figure 3 for an example with one node). The third step is to add to this structure those o_j whose lines between them and {B} are contained within at least one circular sector from step 2. The graph-like structure co now consists of all o_j within a circular sector, and thus has two well defined edge points. From these two points the one closer to W is named o_e, and β_D is chosen to be the angle to it.

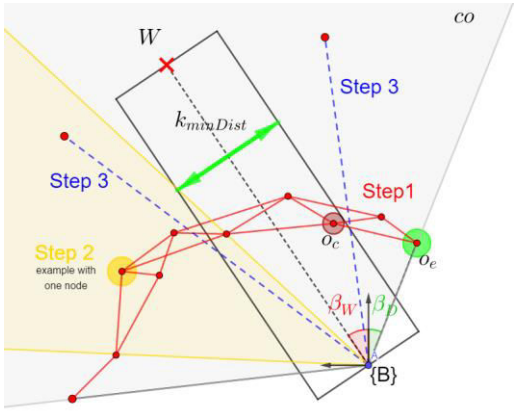

Figure 3. Merging of the obstacle points and selection of the edge point.

2) Turn Left or Turn Right

We can now use β_D and the x-coordinates of o_j to divide all KED_{ij} to those that should avoided by turning left ("go up" in KES) and to those that should be avoided by turning right ("go down" in KES). All KED_{ij} are divided to four sets:

$$KED_{front,-} = \{ KED_{ij} | x_{oj} > 0 \text{ and } \beta_j < \beta_D\},$$
$$KED_{front,+} = \{ KED_{ij} | x_{oj} > 0 \text{ and } \beta_j > \beta_D\},$$
$$KED_{back,-} = \{ KED_{ij} | x_{oj} < 0 \text{ and } \beta_j < \beta_D\},$$
$$KED_{back,+} = \{ KED_{ij} | x_{oj} < 0 \text{ and } \beta_j > \beta_D\}.$$

The closest KED_{ij} in KES that we would like to avoid by turning left has the value

$$KED_{obs,+} = \min(\min(KED_{front,-}), 2\min(KED_{back,+}))$$

and similarly for those KED_{ij} that are avoided by turning right:

$$KED_{obs,-} = \min(\min(KED_{front,+}), 2\min(KED_{back,-})).$$

The multiplier of two of the "back" parts is there because de-emphasizing the obstacle proximity to the rear-half of the robot gives slightly better performance with non-circular robots. This multiplier could be some other small value larger than one as it has a limited impact on the overall behavior.

The control system is constructed so that the robot would generally try to steer left if $KED_{obs,+} > KED_{obs,-}$ and right if $KED_{obs,+} < KED_{obs,-}$. However for free-space navigation we also need methods that do not depend on the proximity of obstacles. The solution chosen here is

$$\omega_{steer} = \omega_{max} \frac{(KED_- - KED_+)}{\min(KED_-, KED_+)},$$

$$KED_{+} = \min(KED_{obs,+}, KED_{default}(1 + |\min(0, \beta_D)|),$$

$$KED_{-} = \min(KED_{obs,-}, KED_{default}(1 + \max(0, \beta_D)),$$

where $KED_{default}$ is some relatively large default value. Effectively, if "far" away from obstacles, the robot will tend to steer towards W (or o_e in case of obstruction), and the smaller $KED_{default}$ is, the closer to the obstacles the robot navigates.

3) Translational Velocity

Many heuristics could be used to set the desired translational velocity, but as will be shown even setting $v_{steer} = v_{max}$ works well enough. Similarly, many heuristics could be used to slow down and stop the robot near (stationary) W, but this will not be addressed in this example controller.

4) Limit to Allowed Velocities

If steering velocities exceed the allowed velocity limits they are capped to the maximum allowed values

$$v_{capped} = \begin{cases} \min(v_{steer}, v_{alwd,+}), if\ v_{steer} > 0 \\ \max(v_{steer},\ v_{alwd,-}), else \end{cases},$$

$$\omega_{capped} = \begin{cases} \min(\omega_{steer}, \omega_{alwd,+}), if\ \omega_{steer} > 0 \\ \max(\omega_{steer},\ \omega_{alwd,-}), else \end{cases}.$$

5) Recovery Mode

As the controller is reactive, local, and memoryless, it is possible that the robot will become stuck. This is detected by checking $|v|$, $|\omega|$, $|v_{capped}|$and $|\omega_{capped}|$. If all these values are below some small threshold k_{RM}, then the robot is considered stuck, as it is basically not moving and is not allowed to start moving in the desired way. In this case, the recovery mode is activated for t_{RM} seconds. In recovery mode sections III.B.2 and III.B.3 are replaced with $v_{steer} = -v_{max}$, $\omega_{steer} = sign(\omega_{steer})\omega_{max}$ and section III.B.4 is run normally to produce v_{capped} and ω_{capped}. If both of these new v_{capped} and ω_{capped} remain below k_{RM}, then the steering velocities are replaced with

$$v_{steer} = \begin{cases} v_{max}, if\ \max(M_1, M_2) > \max(M_3, M_4) \\ -v_{max}, else \end{cases}$$

$$\omega_{steer} = \begin{cases} \omega_{max}, if\ \max(M_1, M_4) > \max(M_2, M_3) \\ -\omega_{max}, else \end{cases}$$

and section III.B.4 is again run normally to produce v_{capped} and ω_{capped}. In other words, the recovery mode first tries to go backwards while turning in the desired direction, and if this is not possible, go in a direction that is generally the safest in KES. In either case, after the recovery mode has run for t_{RM} seconds, normal operation is resumed. This recovery behavior is very rudimentary, and the example controller should be in practice accompanied by a more robust planner, e.g. something from the algorithm family based on RRT [15], which can be given control if the robot gets stuck.

C. Robot Acceleration Twist

A very simple proportional control scheme with acceleration limits was used to produce the final allowed robot twist,

$$\Phi_c = sign(\omega_{capped} - \omega)\min(\Phi_{capped}, k_a|\omega_{steer} - \omega|),$$
$$a_c = sign(v_{capped} - v)\min(a_{capped}, k_a|v_{steer} - v|).$$

1) Wheel Velocity Commands

The acceleration twist could be realized with some torque control method like [9], but motor controllers often accept only velocity values. In such a case, the new desired wheel velocities can be calculated with

$$\omega_{w1,t+1} = \omega_{w1,t} + \Delta t(a_c - y_{w1}\Phi_c)/R_{w1},$$
$$\omega_{w2,t+1} = \omega_{w2,t} + \Delta t(a_c - y_{w2}\Phi_c)/R_{w2},$$

where $\omega_{wn,t}$is the current rotational velocity of wheel n and Δt is the control cycle length.

IV. Simulations

A. Simulation Setup

In the example controller KED_{min}, KED_{max} and $KED_{default}$ are the parameters with the most room for meaningful experimentation as all the others are either model parameters, which should be selected accurately or pessimistically, or some soft behavioral limits that arise from operational preferences. A deeper study on how to choose KED_{min}, KED_{max} and $KED_{default}$ is subject to further work, but as seen in the simulations, even simple heuristics produce acceptable results.

The simulations shown in this section were done with the CoppeliaSim simulator [11] using the Newton [12] physics-engine plugin. The control cycle was 50ms and the internal time-step of the dynamics engine was the default 5ms. Four simulated scenarios with a twisting path were run with three different robots. The obstacles were sensed with a simulated 360 degree 2D LIDAR. Each wheel motor was capable of producing a 70N force with ground contact. The motors were given velocity commands as in section III.C.1.

The common robot parameters were $f_1 = f_2 = 50\ N$, $v_{max} = 1\frac{m}{s}$, $\omega_{max} = 2\frac{rad}{s}$, $a_{max} = 2\frac{m}{s^2}$, $\Phi_{max} = 2\frac{rad}{s^2}$, $M = 76$, $N = 90$, $\Delta t = 50ms$, $k_{RM} = 0.05$, $t_{RM} = 1s$, $k_a = 10$, $k_{pathDist} = 3m$, $KED_{default} = 60J$, $KED_{min} = 5J + |E_v| + |E_\omega|$, and $KED_{max} = 20J + |E_v| + |E_\omega|$. Note that KED_{min} and KED_{max} increase when the velocity of the robot increases. This simple heuristic was chosen because physical disturbances generally increase when velocities increase.

For the large rectangular 2m*1.5m robot, and the large circular one with a radius of 0.75m; $R_1 = R_2 = 0.22m$, $y_1 = 0.57m$, $y_2 = -0.57m$, $m = 40kg$, and $I = 21kg\ m^2$.

For the small rectangular 0.6m*0.4m robot; $R_1 = R_2 = 0.05$, $y_1 = 0.15m$, $y_1 = -0.15m$, $m = 15kg$, $I = 10kg\ m^2$, and $KED_{default} = 20J$ (during the second run).

In each scenario the environment was identical and the waypoint W moved on the same (thick blue) path that was (poorly) constructed by some global planner. The path {B} made is the green line, and the wheel paths are the thin blue lines.

B. Simulation Results

The route of the large circular robot was fairly smooth both in terms of curvature and velocities as seen in Figure 4. In this case and all the following ones, W was behind obstacles most of the time.

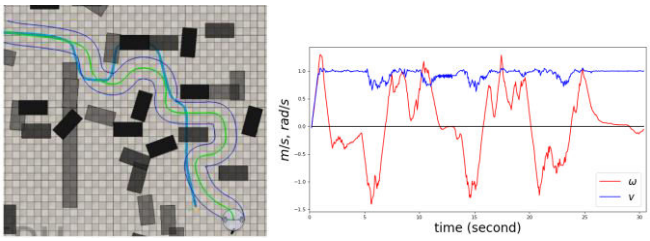

Figure 4. Left, large circular robot in a corridor. Right, its velocities.

With the large rectangular robot the outline of the robot was drawn every 0.25s to better visualize the taken route (see Fig. 5). The recovery mode was activated twice, first after the first narrowing, and then on the L-turn. The corridors were admittedly set-up so that the controller could successfully navigate the robot through them, but this scenario nevertheless demonstrated that the controller can control large, non-circular robots in fairly demanding environments.

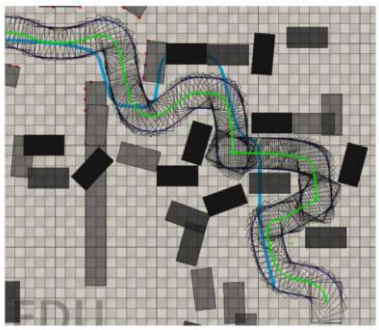

Figure 5. Large rectangular robot in a corridor.

The small robot was run twice: once with $KED_{default} = 60J$ and a second time with $KED_{default} = 20J$. In Fig. 6 the effect of changing $KED_{default}$ is clearly visible. With the higher value the robot tended to keep more free space around it. With the lower value the robot tended to take straighter routes closer to the obstacles.

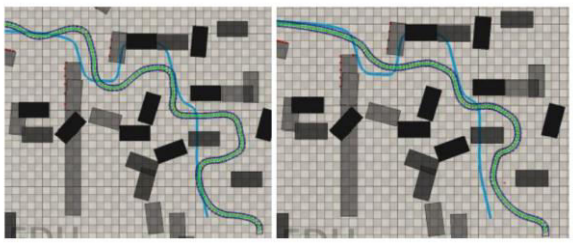

Figure 6. Left, $KED_{default} = 60J$. Right, $KED_{default} = 20J$.

In summary, the simulations show that the example controller can successfully navigate robots with slow dynamics through narrow corridors despite quite a poor path from the global planner.

V. Test With a Real Robot

The controller was further tested with a real robot in a simple scenario with static W, mainly for testing the performance with a low-cost computer. The robot was akin to the small simulated one, with a Rasberry Pi 4 Model B computer, and Hokuyo UST-20LX-H01 270 degree LIDAR. The simplified test was run multiple times with N=120 obstacle points, varying M between 30-120, $v_{max} = 0.125\frac{m}{s}$, and a 200ms control cycle. The visible behavior in each run was very similar (see Figure 7), but not identical, due to at least some extent to the noise in the unfiltered LIDAR measurements.

Figure 7. Simple test run with a real robot.

The example controller was written in Python, with only rudimentary optimization, and most notably, as a single-threaded program running on one of the four cores of the CPU. Table 1 displays the percentage *increase* of the *total* average CPU load when running the example controller with a 500ms control cycle with different M and N. The data in Table 1 implies that just with proper parallelization, the controller could, for example, operate at a 50ms control cycle with M=120 and N=240, using roughly 70% of the very affordable CPU's processing power. Properly optimized C implementation would further increase the performance, and GPU utilization much more so. Likewise, various heuristics and other improvements could be used to reduce the amount of processed (b_i, o_j) pairs, e.g. reducing M if all o_j are relatively far away from the robot.

TABLE I. Increase In the Total CPU Load With a 500ms Control Cycle

	N = 60	N = 120	N = 180	N = 240	N = 300	N = 360
M=30	+1,4%	+2,7%	+3,6%	+4,0%	+4,3%	+5,8%
M=60	+2,4%	+3,1%	+3,7%	+5,1%	+5,7%	+6,9%
M=120	+2,5%	+4,2%	+5,8%	+7,0%	+8,1%	+10,0%

VI. Conclusion

This paper introduced distance in the kinetic energy space as a metric for collision proximity, and provided a simple computationally inexpensive example obstacle avoidance algorithm that utilizes it. Further work includes refining the example controller, especially the selection of KED_{min} and KED_{max}, integrating KED to well-established methods such as DWA [6], and extending the metric to robots with different kinematics.

Acknowledgments

The authors would like to thank the Finnish Cultural Foundation, the University of Oulu Graduate School (UniOGS) and Infotech Oulu for making this research possible. The authors would also like to thank BSc Timo Mäenpää for his help with section V.

REFERENCES

[1] M. Hoy, A.S. Matveev, and A.V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," Robotica, vol. 33, 2015, pp. 463-497.

[2] D.Q. Mayne and S. Raković, "Model predictive control of constrained piecewise affine discrete-time systems," International Journal of Robust and Nonlinear Control, vol. 13(3-4), 2003, pp. 261-279.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," International Journal of Robotics Research, vol. 5(1), 1986, pp. 90-98.

[4] R.B. Tilove, "Local obstacle avoidance for mobile robots based on the method of artificial potentials," IEEE International Conference on Robotics and Automation, Cincinnati, OH, USA, 1990.

[5] S.S. Ge and Y.J. Cui, "Dynamic motion planning for mobile robots using potential field method," Autonomous Robots, vol. 13(3), 2002, pp. 207-222.

[6] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," IEEE Robotics & Automation Magazine, vol. 4(1), 1997, pp. 23-33.

[7] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," International Journal of Robotics Research, vol. 17(7), 1998, pp. 760-772.

[8] J. Minguez, L. Montano, J. Santos-Victor, "Abstracting vehicle shape and kinematic constraints from obstacle avoidance methods," Autonomous Robots, vol. 20(1), 2006, pp. 43-59.

[9] V. Pitkänen, V. Halonen, A. Kemppainen, and J. Röning "Path following controller for Differentially Driven Planar Robots with Limited Torques and Uncertain and Changing Dynamics." *Robotics and Automation (ICRA), 2019 IEEE International Conference on,* Montreal, 2019.

[10] S. Ebetiuc, S. Haraldw, "Applying Differential Geometry to Kinematic Modeling in mobile Robotics", *2005 WSEAS Int. Conf. on Dynamical systems and control, Italy*, 2005, pp. 106-112.

[11] http://www.coppeliarobotics.com/

[12] http://newtondynamics.com/forum/newton.php

[13] J-L. Blanco, J. Gonzalez-Jimenez, and JA. Fernandez-Madrigal, " Extending obstacles avoidance methods through multiple parameter-space transformation" Autonomous Robots, vol. 24, no. 1, (2008) pp. 29-48.

[14]] M. Jaimez, J-L. Blanco, and J. Gonzalez-Jimenez, "Efficient Reactive Navigation with Exact Collision Determination for 3D Robot Shapes." International Journal of Advanced Robotic Systems. May 2015.

[15] S.M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in Proceedings of the International Conference on Robotics and Automation, Detroit,USA,1999.