Appendix

A. *Virtual Obstacle*

Considering, available image processing algorithms can infer approximate size of a detected object, let us assume there are several closely spaced obstacles with no passage space in between. The collection of obstacles can be grouped together to form a larger obstacle. Considering the centroid of each individual obstacle as a vertex of a planar polygon, the centroid of the constructed polygon is the centroid of the set of obstacles. The computed point (centroid) forms the centre of the virtual obstacle, (x_{obs}, y_{obs}). Let, the largest line segment joining the centroid of the set of obstacles and any of the centroids of the individual obstacles be denoted as r_1. Let, the line segment joining the corresponding obstacle centroid and its farthest vertex be termed as r_2. Then, $r_1 + r_2$ forms the radius r_{obs} of the virtual obstacle. A single obstacle of arbitrary shape can also be treated like an irregular non-intersecting polygon and the foregoing procedure can be followed. Figure 1 explains the mechanism pictorially.

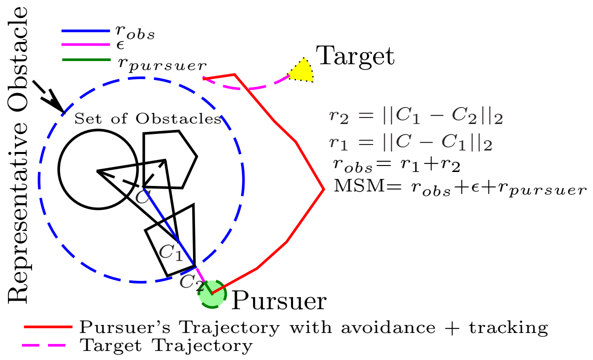

Fig. 1. A virtual obstacle can be created from closely spaced obstacles of arbitrary shapes.

B. *Intent Assessment*

Suppose, latest poses (of target/obstacle) are available at each sampling time, δt. At t, the time-to-intercept is $T_{int} - t$, which is configured to comprise of a maximum of $n = 1 + \frac{T_{int}}{\delta t}$ datapoints in a planning iteration. With each new measurement at δt, the evolution of data can be expressed in the form of a polynomial function of time, corresponding to a polynomial of maximum degree $(n - 1)$.

The procedure of extrapolation by Neville algorithm generates a polynomial of 0^{th} order when only one data is available at $t = 0$. Then it proceeds by adding a degree to the polynomial when a new pose is read from the sensors. As the planning horizon shrinks, the maximum degree of polynomial constructed can remain same, thereby reducing the sampling time and possibilities of constraint violation. Here, t represents the independent variable and x and y are the dependent variables. Thus two polynomials are constructed in each planning iteration, one for x and the other for y (states may belong to target/obstacle(s)). It may be recalled again, the parametric form of target and obstacle motion are used only for intent assessment at a future time, which can be either the time-to-intercept or a time-to-collide. For target, only the final states at T_{int} are utilized from the extrapolation. Initial guess for constructing the polynomials is usually inaccurate. But as new measurements are obtained, the intent of the moving entity can be conceived better and the polynomial represents a better approximation of the actual trajectory (which is not-arbitrary, but unknown). Eventually, the accuracy of the state estimation at some future instant also increases. Dataflow of the Neville-Aitken algorithm proceeds in the following manner (see Table I).

Table I . Neville's Algorithm .

Iteration	Independent variable	Measured variable	Polynomial D(1)	Polynomial D(2)	Polynomial D(3)
0	$t = 0$	$x_0 = f(0) = P_0$			
			P_{01}		
1	$t = \delta t$	$x_1 = f(\delta t) = P_1$		P_{012}	
			P_{12}		P_{0123}
2	$t = 2\delta t$	$x_2 = f(2\delta t) = P_2$		P_{123}	
			P_{23}		
3	$t = 3\delta t$	$x_3 = f(3\delta t) = P_3$			

$$P_{ij} = \frac{t-t(j)}{t(i)-t(j)} f(t(i)) + \frac{t-t(i)}{t(j)-t(i)} f(t(j)), \qquad P_{ijk} = \frac{(t-t(k))P_{ij} - (t-t(i))P_{jk}}{t(i)-t(k)}$$

Thus, at the i^{th} instant in any planning iteration, the resultant polynomial is $P_{012\ldots i}$, which is then evaluated at a future instant (for example, T_{int} for target) to predict states of the target and the obstacles at that future instant.

C. Solution to Optimal Control Problem

Equations in (9) represent a class of boundary value problems, having the form $\dot{s} = \mathfrak{f}(t, s)$, where s is the 4-element state vector $[x, y, \lambda_1, \lambda_2]$. The boundary constraint functions are of the form $g(s_i(0)) = a$ and $g(s_i(T_{int})) = b, \forall i \in \{1, 2, 3, 4\}$, where, a and b are real numbers and the independent variable is time t, ($t \in [0, T_{int}]$). The existence-uniqueness of such problems have been investigated in ref. [42] using Perov's comparison theorem.

In this case, the boundary values of the co-states are unknown. In fact, the co-states are free to assume any numerical value satisfying the constraint functions, $g(s_i(0))$ and $g(s_i(T_{int}))$. It is justified to reason that, for each unique pair of states $(x(t), y(t))$ there is *at least* a pair of $(\lambda_1(t), \lambda_2(t))$, $\forall t \in [0, T_{int}]$. However, it can be proved that there cannot exist more than one sequence of (λ_1, λ_2) in the said interval, corresponding to each unique solution, $(x(t), y(t))$ defined in that interval. In absence of a closed form solution to (9), an approximate numerical solution can be computed that satisfy (2). Then, tracing back to the start of this approximate sequence, one can find a set of $(\lambda_1(0), \lambda_2(0))$ corresponding to the solution generated by the numerical solver. The objective it to determine, if starting with that approximate initial values of the co-states and the known initial values of the states, it is possible to arrive at two different solutions (with same initial and terminal values only) in the above mentioned interval. The short answer is 'no' and here is the reason, why.

Let us recall two fundamental assumptions:

1. The co-states, $\lambda_i(t)$ are continuous and bounded functions of time; this is true by definition.
2. There is a unique set of $(x(t), y(t))$ that satisfy the boundary conditions in the closed interval $[0, T_{int}]$ (if there are a number of solutions, $(x(t), y(t))$, the following discussion applies to each, separately).

In ref. [43], a detailed explanation has been provided using Picard's theorem, which points out the validity of second condition. Now, starting with the said set of initial values of states and co-states, if the right hand side of (9) can be shown to be continuous, bounded and Lipschitz, in the closed interval, $[0, T_{int}]$, then it can be proved by applying *continuous dependence theorem* [43] on the initial conditions that a solution, $(x(t), y(t), \lambda_1(t), \lambda_2(t))$, starting inside the closed interval, $[0, T_{int}]$ will evolve *uniquely* over the said closed interval. This narrows down the task to verification of the Lipschitz condition by checking the boundedness property of the partial derivatives of the expressions in the right hand side of (9).

Let us define $s(t)$:=$[x(t)\ \ y(t)\ \ \lambda_1(t)\ \ \lambda_2(t)]^\top$ and rewrite (9) as given below (A1).

$$\dot{s}(t) = \mathfrak{f}(t, s(t)) = [\dot{x}(t)\ \ \dot{y}(t)\ \ \dot{\lambda}_1(t)\ \ \dot{\lambda}_2(t)]^\top \tag{A1}$$

Here, $\mathfrak{f}(t, s(t))$ is continuous in $s(t)$ and bounded, because, the augmented state trajectories are continuous and bounded functions of time. Now, the partial derivatives of $\mathfrak{f}(t, s(t))$ with respect to $s(t)$ are computed as $\Delta\mathfrak{f}_i$=$\left[\frac{\partial\mathfrak{f}_i}{\partial x(t)}\ \frac{\partial\mathfrak{f}_i}{\partial y(t)}\ \frac{\partial\mathfrak{f}_i}{\partial\lambda_1(t)}\ \frac{\partial\mathfrak{f}_i}{\partial\lambda_2(t)}\right]^\top$, $i \in \{1, 2, 3, 4\}$. For brevity, let us look at one of the partial derivatives, $\Delta\mathfrak{f}_1$.

$$\Delta\mathfrak{f}_1 = \left[0\ \ 0\ \ -\lambda_2^2\left(\frac{v_{max}}{(\lambda_1^2+\lambda_2^2)^{1.5}} + \frac{w_v}{(\lambda_1^2+\lambda_2^2)^2}\right)\ \left(\frac{v_{max}(\lambda_1\lambda_2 - (\lambda_1^2+\lambda_2^2))}{(\lambda_1^2+\lambda_2^2)^{1.5}} - \frac{2w_v\lambda_1\lambda_2}{(\lambda_1^2+\lambda_2^2)^2}\right)\right]^\top \tag{A2}$$

Each non-zero element of $\Delta\mathfrak{f}_1$ can be expressed as a rational function, whose denominator polynomial has a greater degree than that of the numerator, which implies, the partial derivative with respect to $s(t)$ has a finite limiting value, and that the function is bounded. The process is same for other partial derivatives. The Lipschitz constant K can be computed as $[\hat{K}_1\ \ \hat{K}_2\ \ \hat{K}_3\ \ \hat{K}_4]^\top$, where, $\hat{K}_i := sup|\Delta\mathfrak{f}_i|$.

The summary is that, a pursuer starting at some point with a set of initial heading and velocity cannot follow two different routes (route implying state and control vectors) to interception. For more than one solution of the optimal control problem, the initial states and co-states must be different for each solution, which is why the numerical solver generates one of the possible solutions depending on the initial conditions.

D. Client-Server and Optimization Algorithms

Algorithm 1: Client

```
Activate: $UDP\_Receiver$, $UDP\_Sender$
Initialize $T_{int}, T_{update}, tol, current\_time$
$i \leftarrow 0, flag \leftarrow 0$
Start: while (1)
if $current\_time \geq T_{update}$ then
    $(x_{tar}(0), y_{tar}(0)) \leftarrow (x_{cam}(i), y_{cam}(i))$
    $(x(0), y(0)) \leftarrow (x_{enc}(i), y_{enc}(i))$
    $T_{int} \leftarrow T_{int} - current\_time$
    Compute: $optimal(x, y), (v, \theta)$
    $dist := \|(x - x_{tar}, y - y_{tar})\|_2$
end if
if $dist \leq tol$ then
    $flag \leftarrow 1$
end if
$UDP\_DataSent \leftarrow (x, y, v, \theta, T_{int}, flag)$
if flag $= 0$ then
    $current\_time \leftarrow 0$
    $i \leftarrow i + 1$, goto Start
end if
close
```

enc : encoder.
cam : camera.
T_{update} : Planning horizon update rate.
tol : Tolerance value of closeness with target.

Algorithm 2: Server

```
Activate: Robot
Initialize: $T_{update}, T_{idle}, t$
$i \leftarrow 1, flag \leftarrow 0$
Start: while (1)
$CurrentPose \leftarrow (x_{enc}(i), y_{enc}(i))$
Open: $UDP\_SendSocket$
$SendBuffer \leftarrow CurrentPose$
$t \leftarrow 0$
Close: $UDP\_SendSocket$
Open: $UDP\_ReceiveSocket$
if $t \geq T_{update}$ then
    goto Reset
end if
if $t \geq T_{idle}$ and $RBuff = NULL$ then
    goto Reset
end if
if $t < T_{idle}$ and $RBuff \neq NULL$ then
    if $flag = 1$ then
        break
    else
        $Robot \leftarrow (v, \theta, T_{int})$
        Reset:
            Close: $UDP\_ReceiveSocket$
            Open: $UDP\_SendSocket$
        $i \leftarrow i + 1$, goto Start
    end if
end if
close
```

$RBuff$: Receive Buffer
T_{idle} : Idle Time

Algorithm 3: Optimization

```
$t \leftarrow current\_time$
**Start:** *while (1)*
Receive $x(t), y(t), x_{tar}(t), y_{tar}(t), x^i_{obs}(t), y^i_{obs}(t), t, i \in [1 \; m]$
*Initial boundary values* $\leftarrow x(t), y(t)$
$T_{int} \leftarrow T_{int} - t, t \leftarrow 0$
**if** Target/obstacle update available **then**
  **goto close**
**end if**
*Estimate:* (Intent Awareness)$x_{tar}(t_1), y_{tar}(t_1), x^i_{obs}(t_1), y^i_{obs}(t_1), t_1 \in (t \; T_{int}]$
*Terminal Boundary values* $\leftarrow x_{tar}(T_{int}), y_{tar}(T_{int}), \Psi(T_{int}) = [0 \;\; 0]^\top$
Assign: $w_v, w_d, w_b$
*Minimize* $J$
*Compute:* optimal $x^*(t_1), y^*(t_1), v^*(t_1), \theta^*(t_1), t_1 \in (t \; T_{int}]$
**close**
```