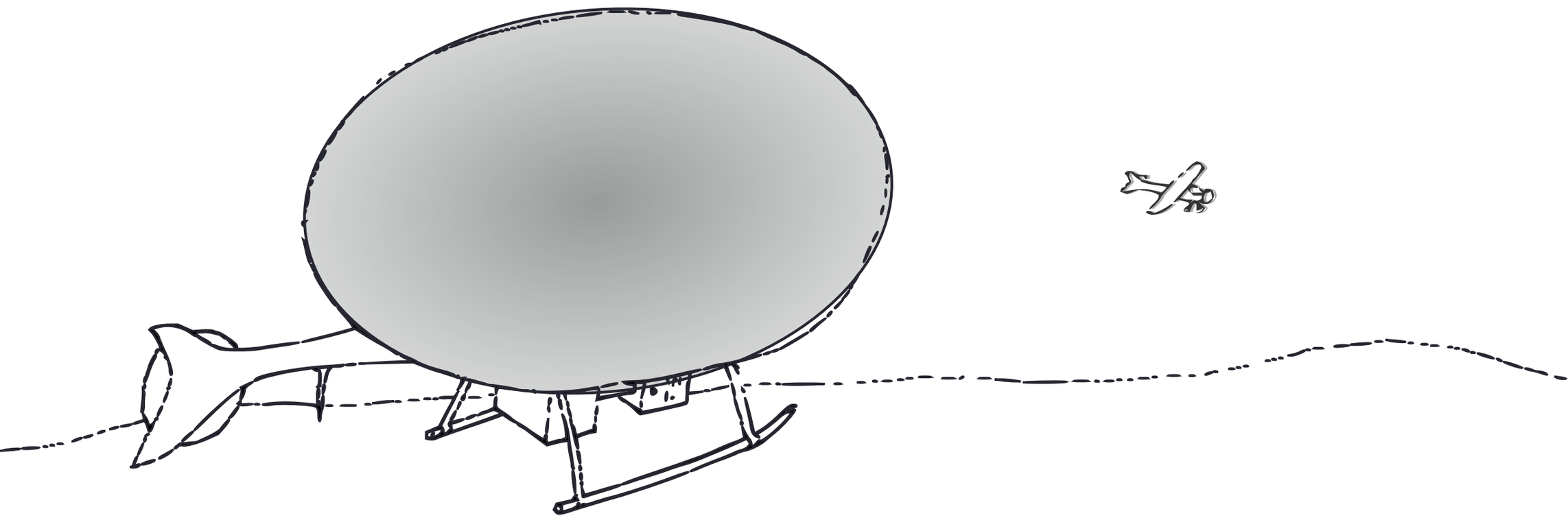
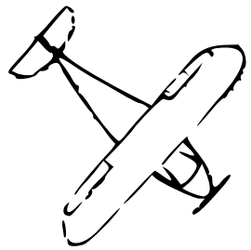
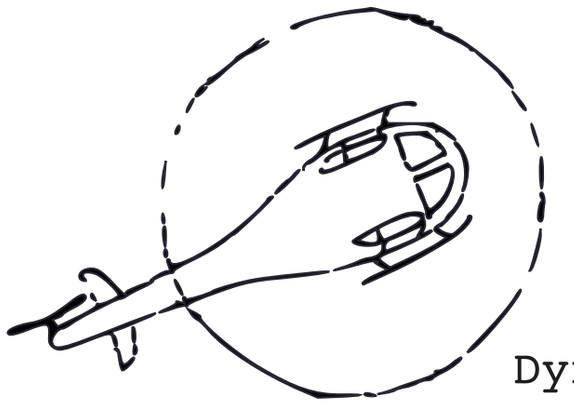


Reactive Obstacle Avoidance

zero to maintainer in twenty(ish) slides

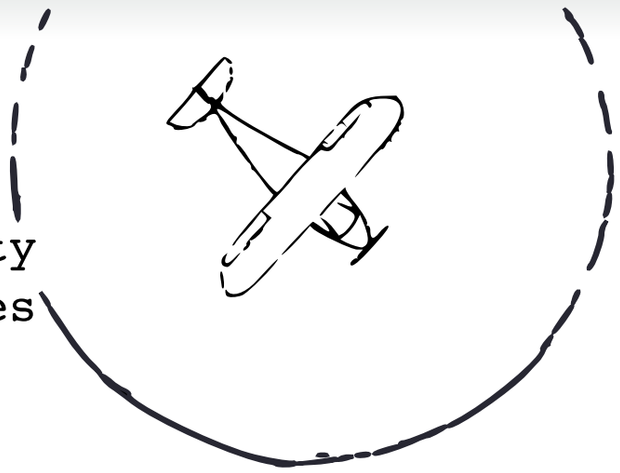
Overview

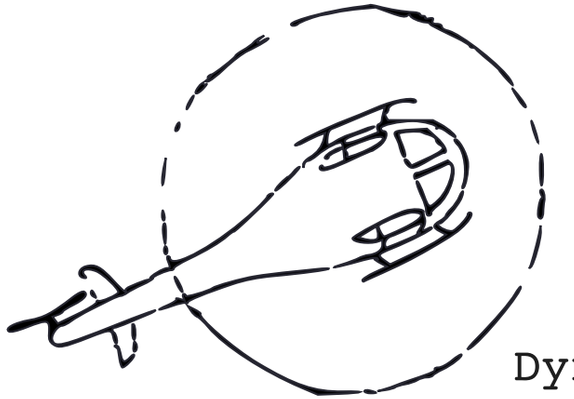




Dynamic avoidance of
non-cooperative obstacles

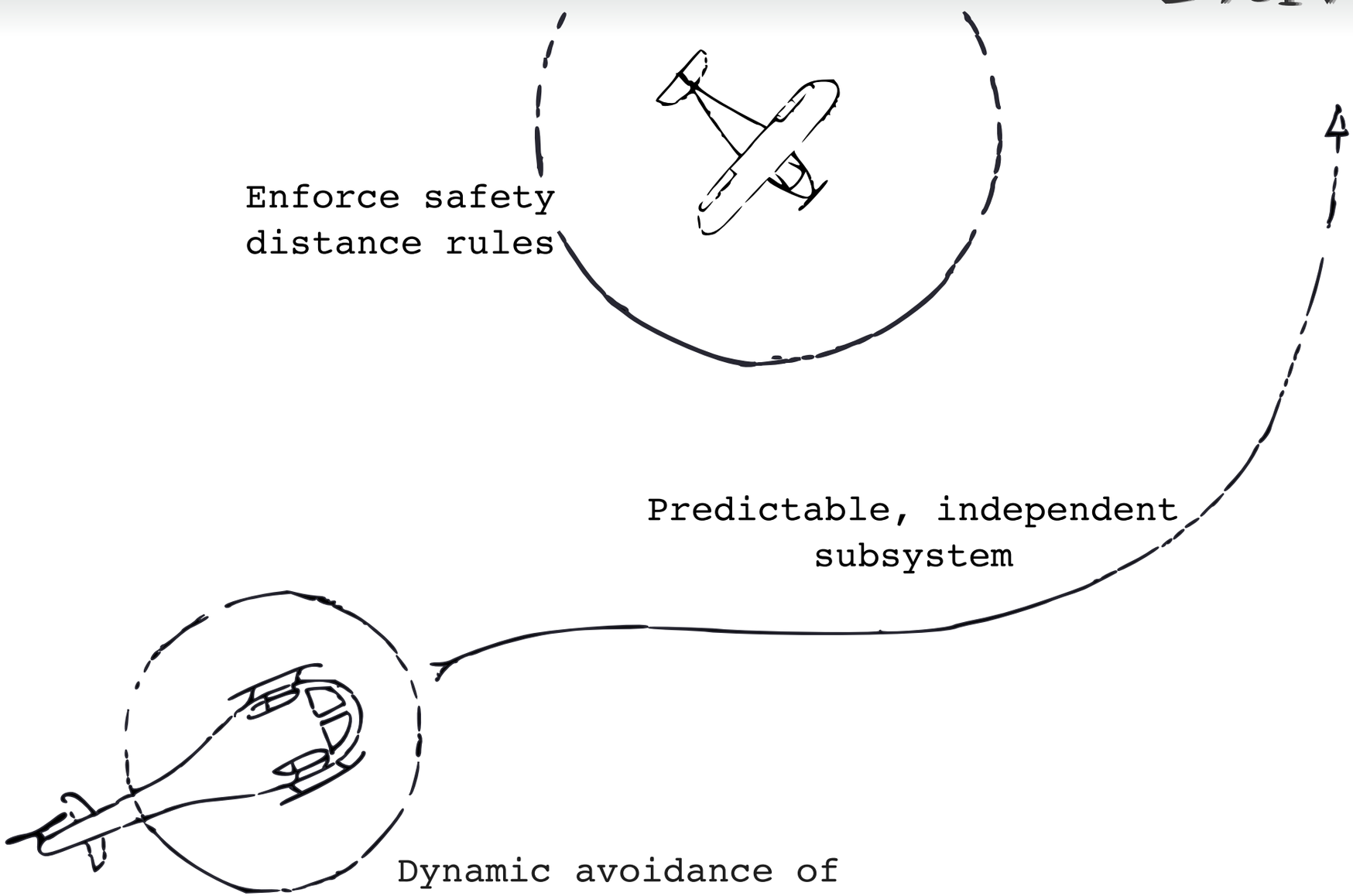
Enforce safety
distance rules





Dynamic avoidance of
non-cooperative obstacles

Overview

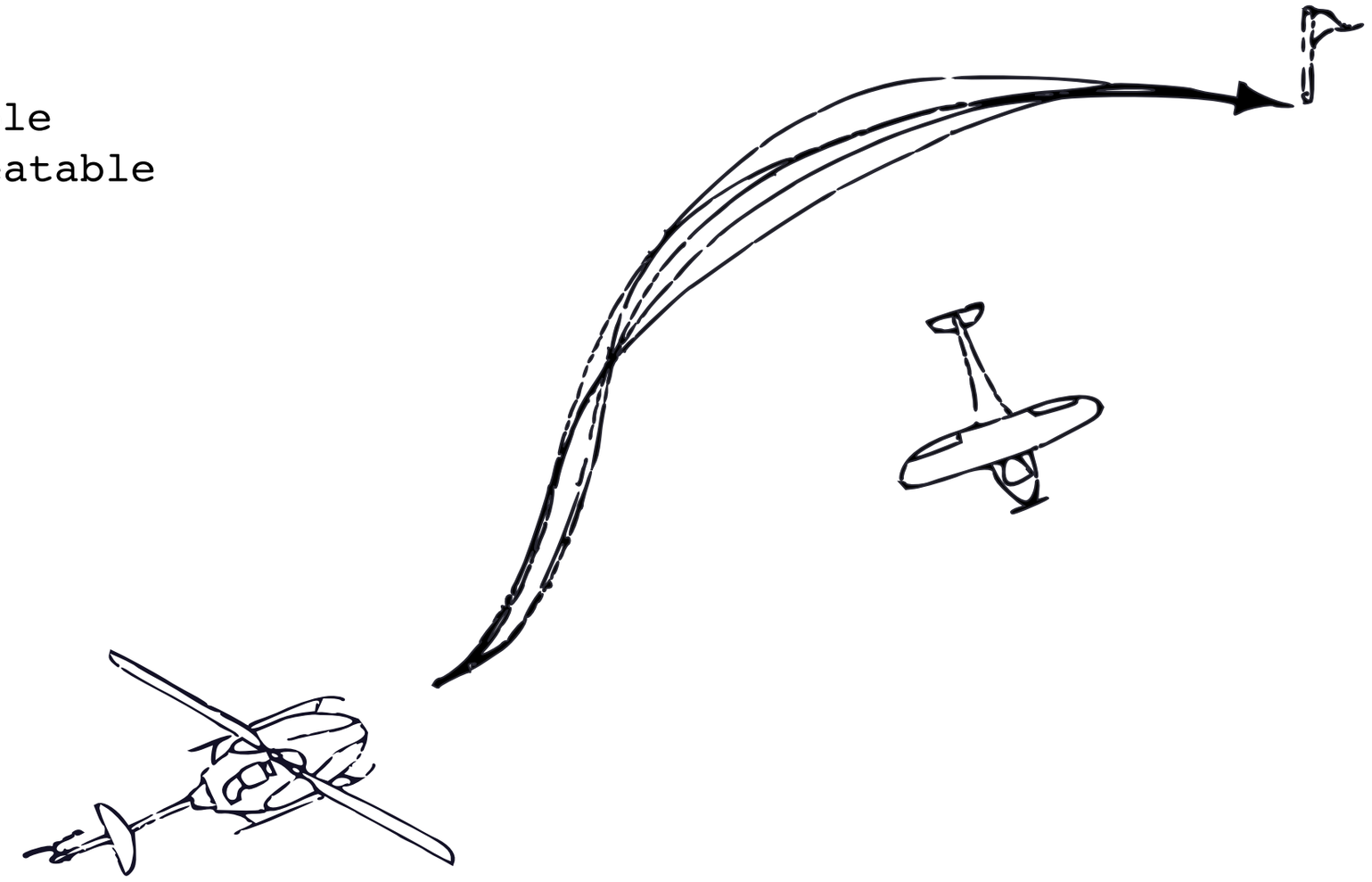

Enforce safety
distance rules
Predictable, independent
subsystem
Dynamic avoidance of
non-cooperative obstacles

- Simple

Obstacle
Here

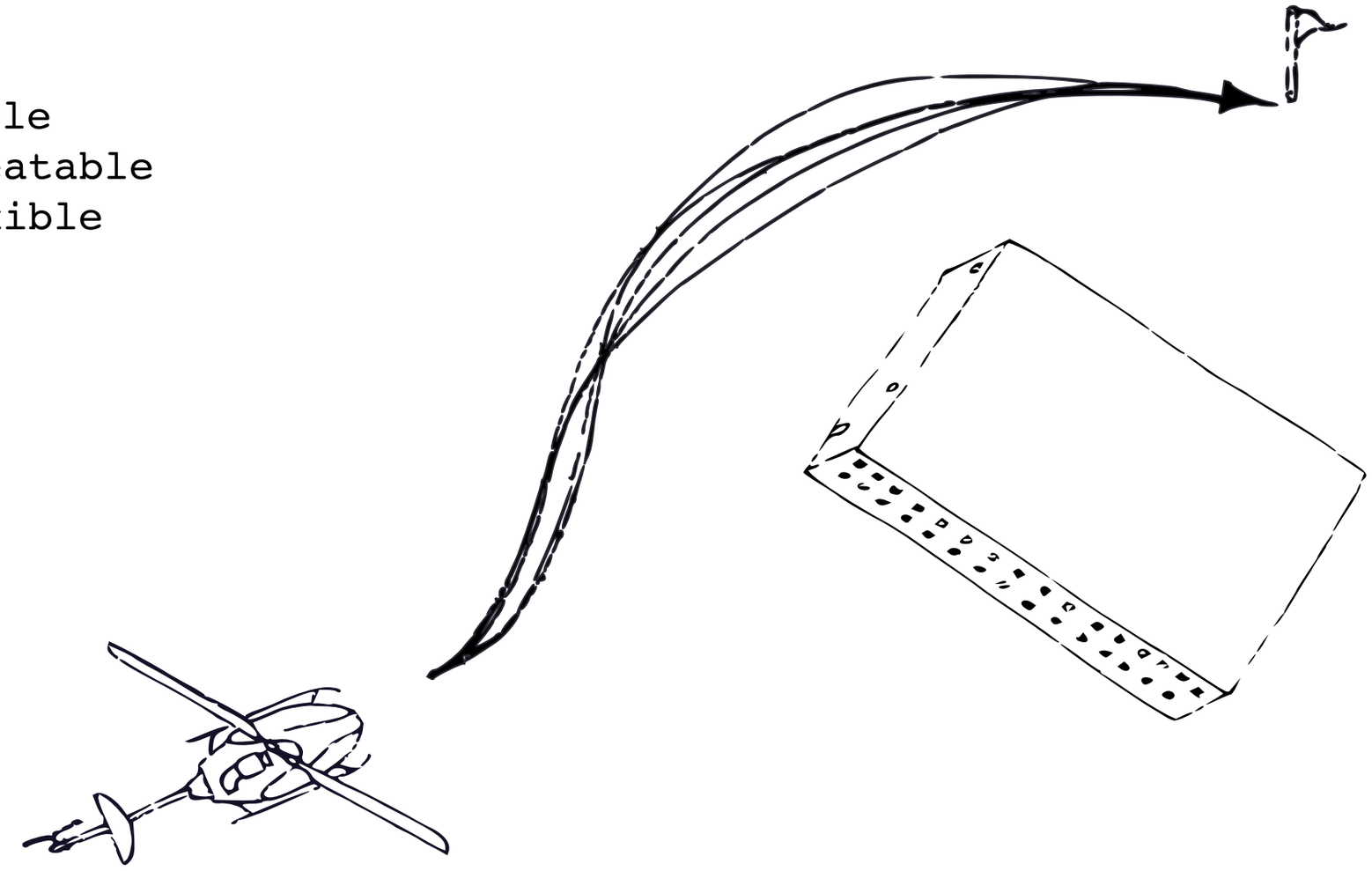
Overview

- Simple
- Repeatable



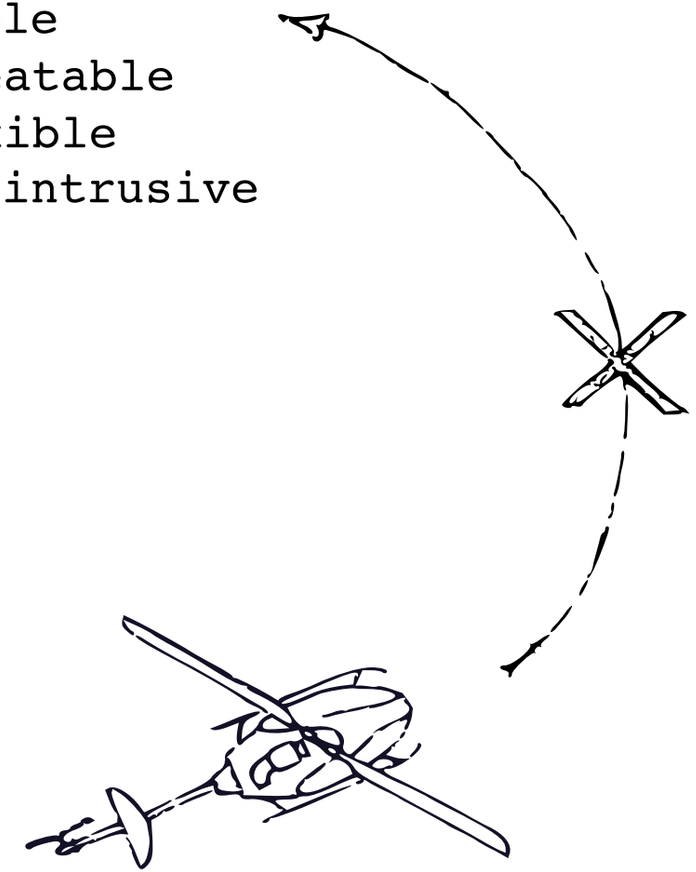
Overview

- Simple
- Repeatable
- Flexible

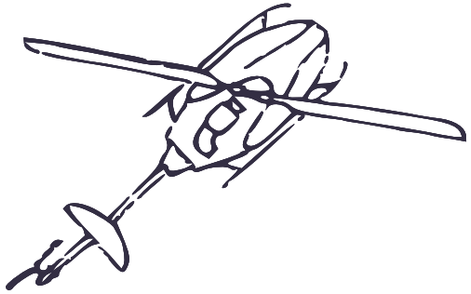


Overview

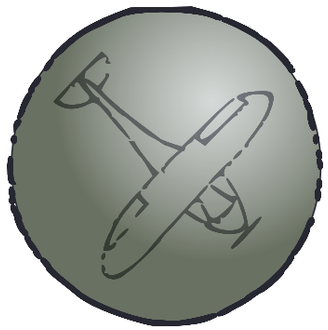
- Simple
- Repeatable
- Flexible
- Non-intrusive

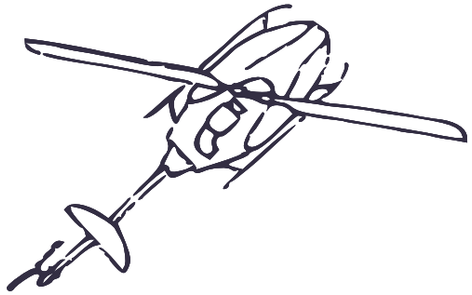


Assumptions



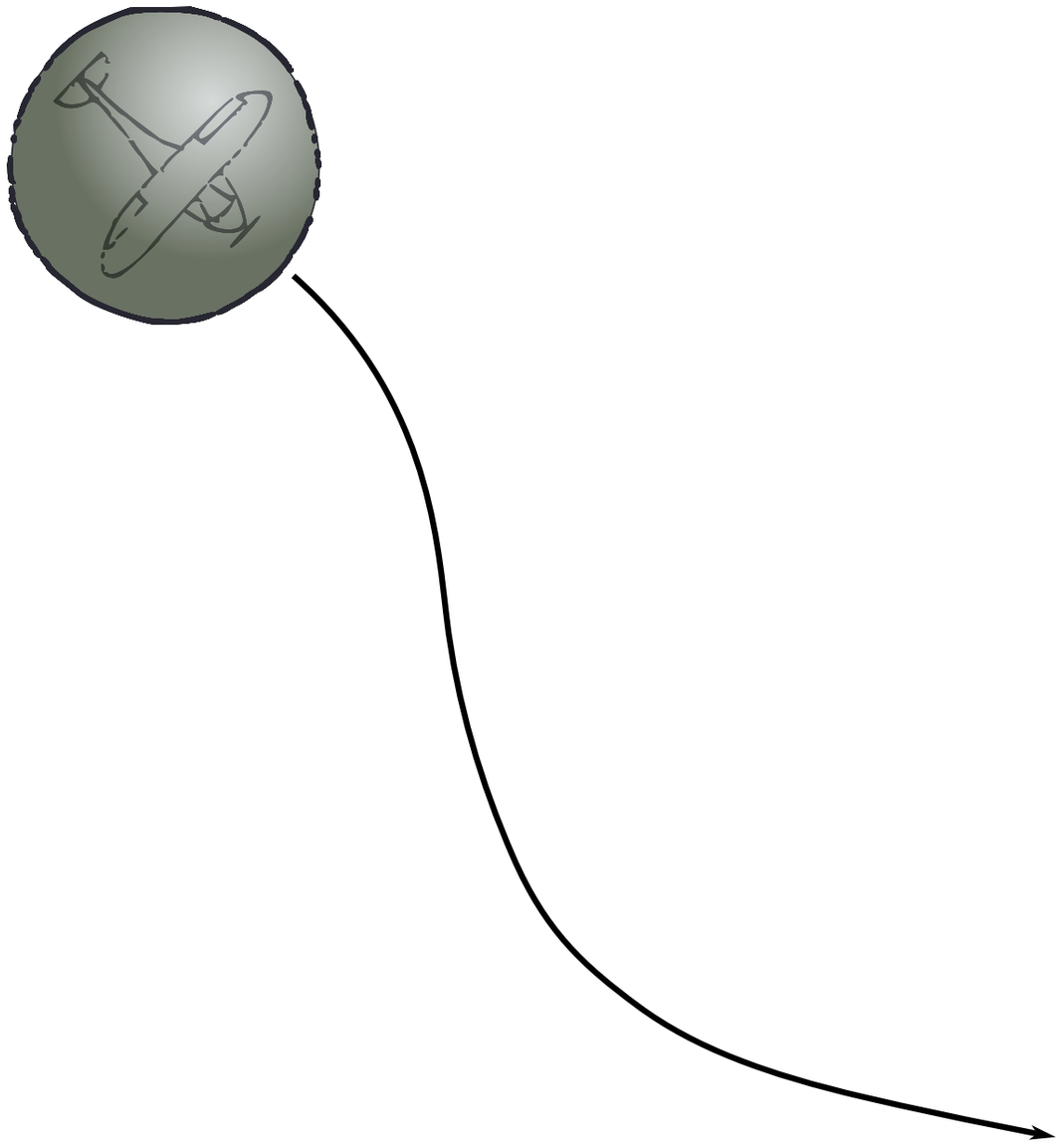
- Objects can be represented as spheres

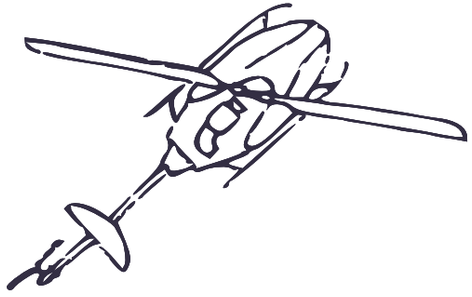




Assumptions

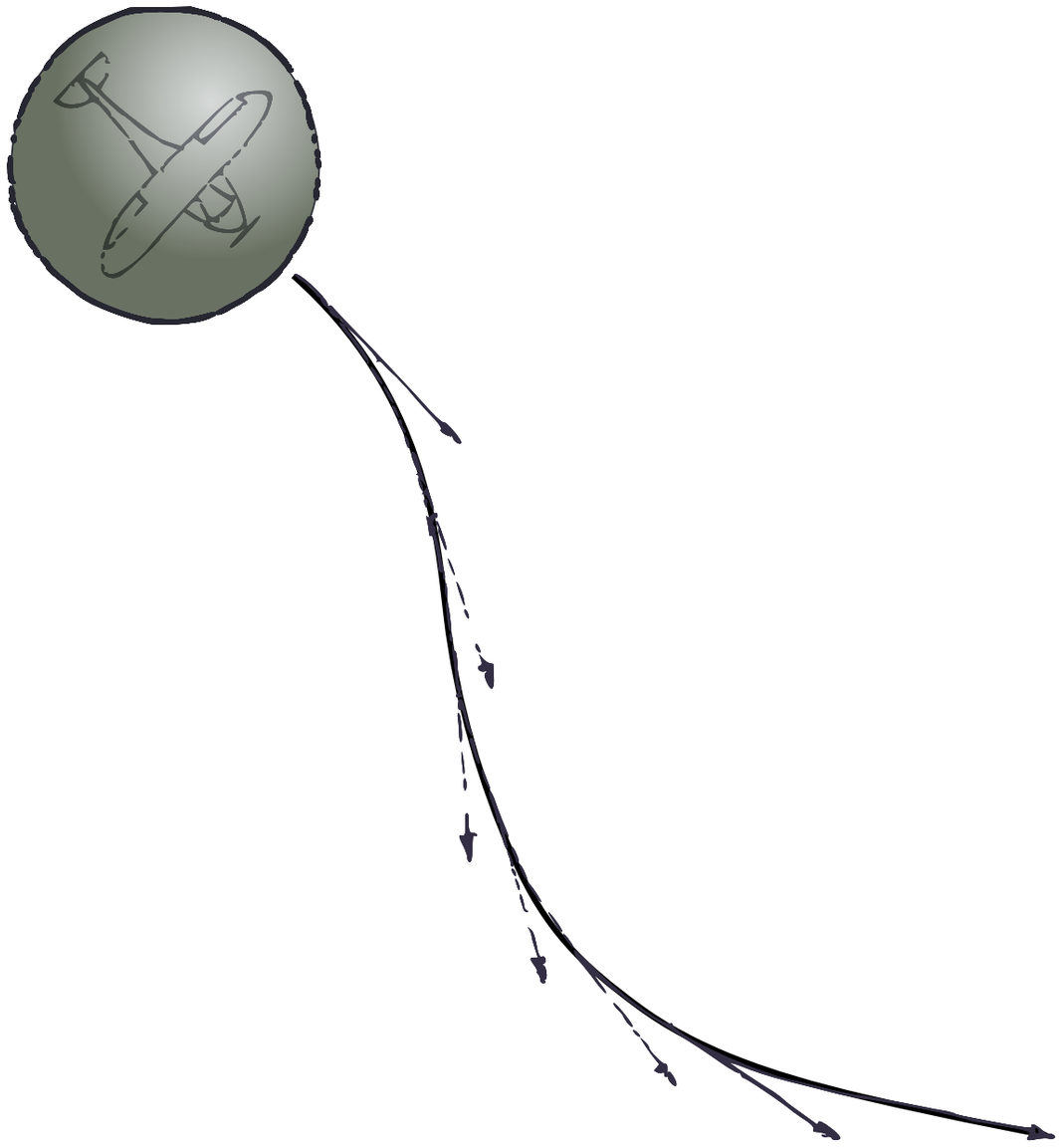
- Objects can be represented as spheres
- Smooth, slow obstacle paths

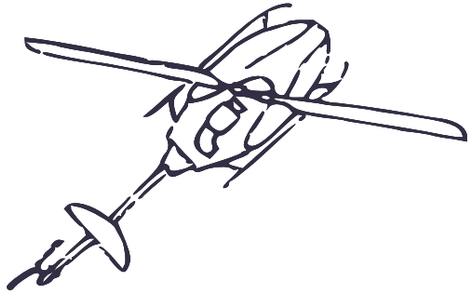




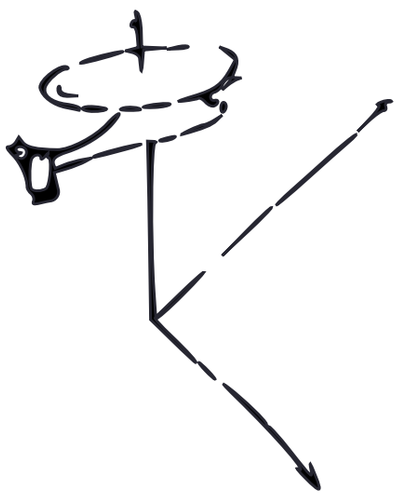
Assumptions

- Objects can be represented as spheres
- Smooth, slow obstacle paths
- Linear velocities accurately predict path

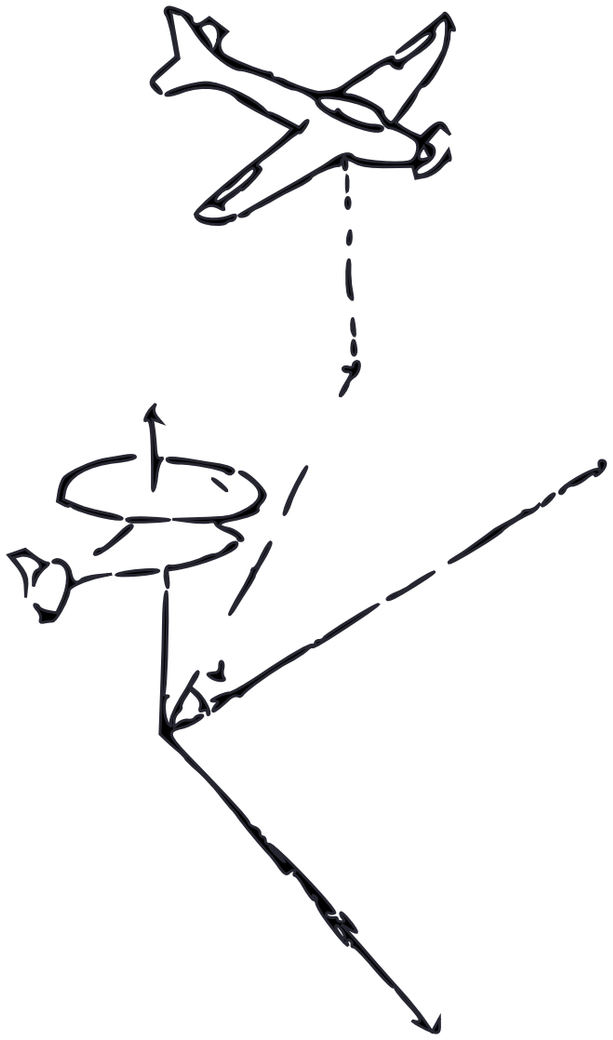




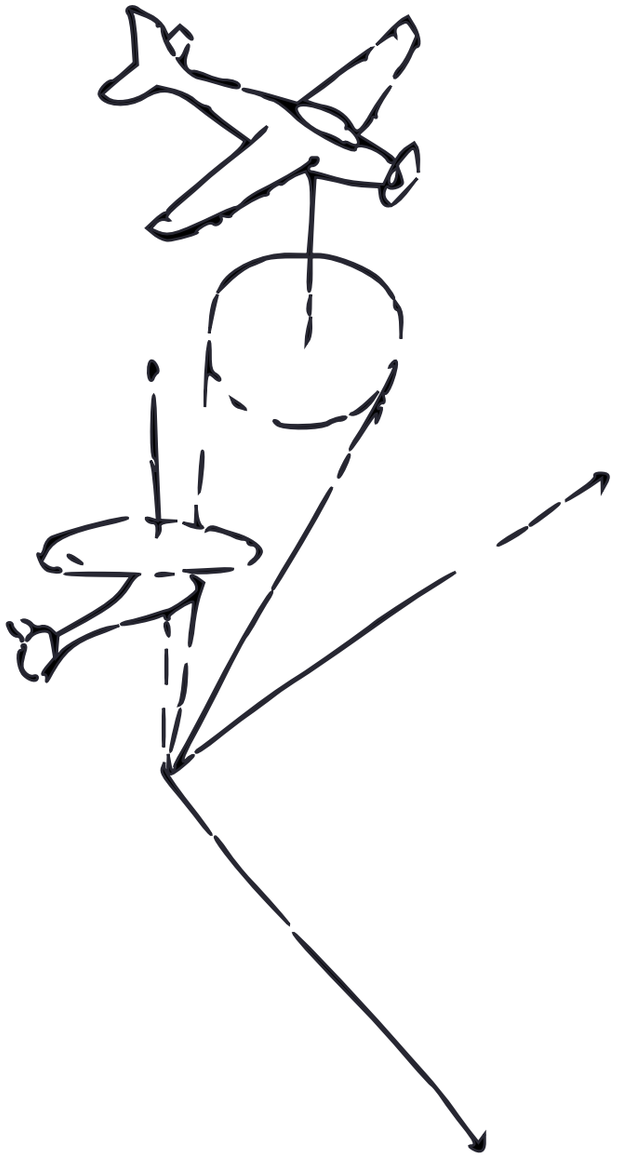
Velocity Obstacles



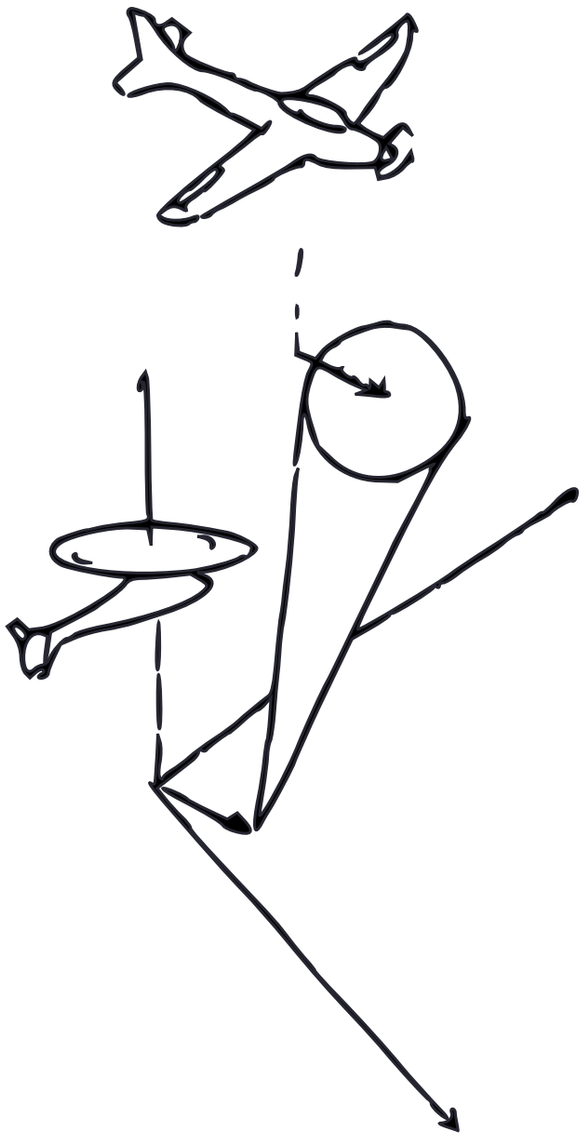
Body-fixed coordinate system



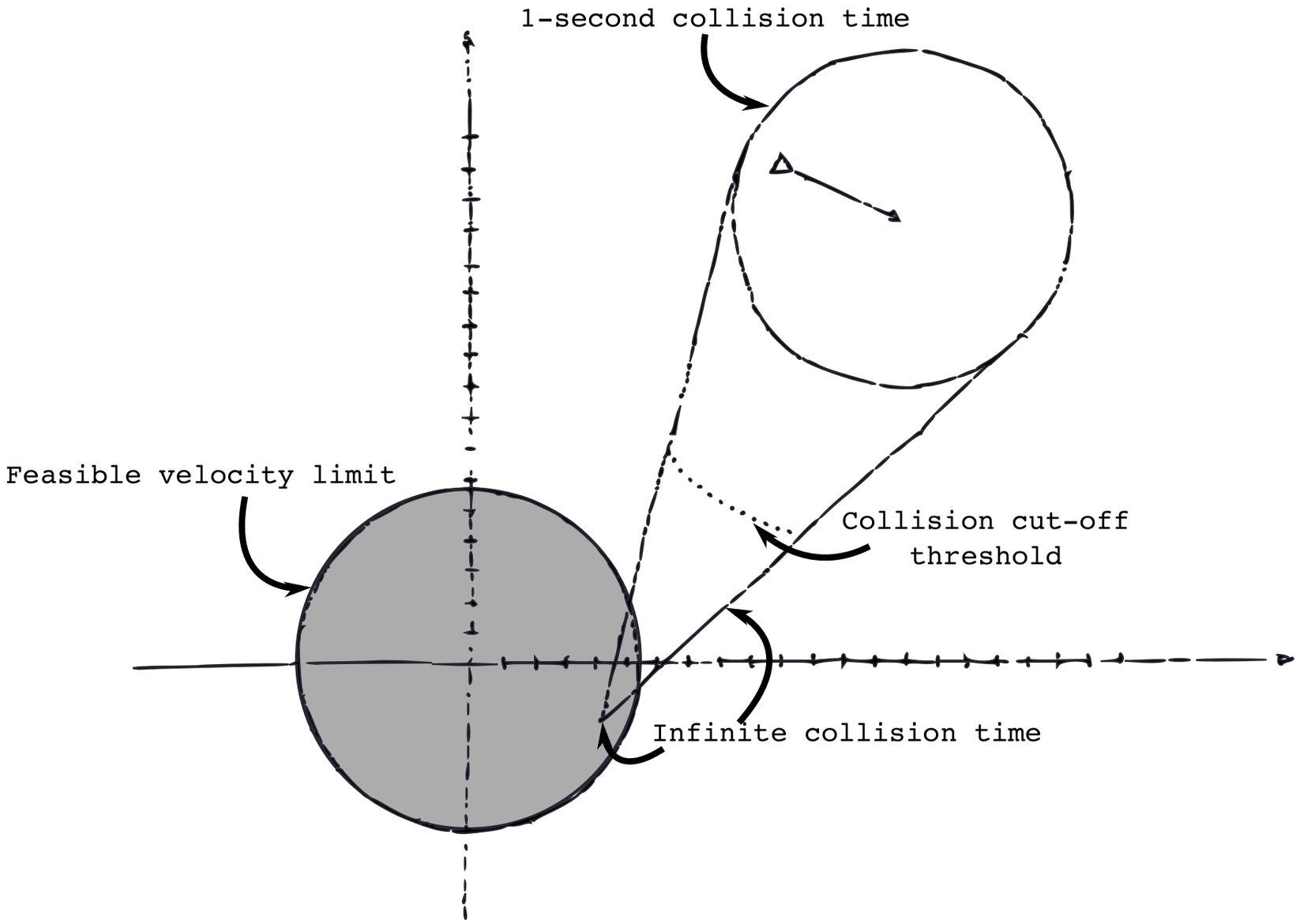
Obstacle localized using its relative position and speed

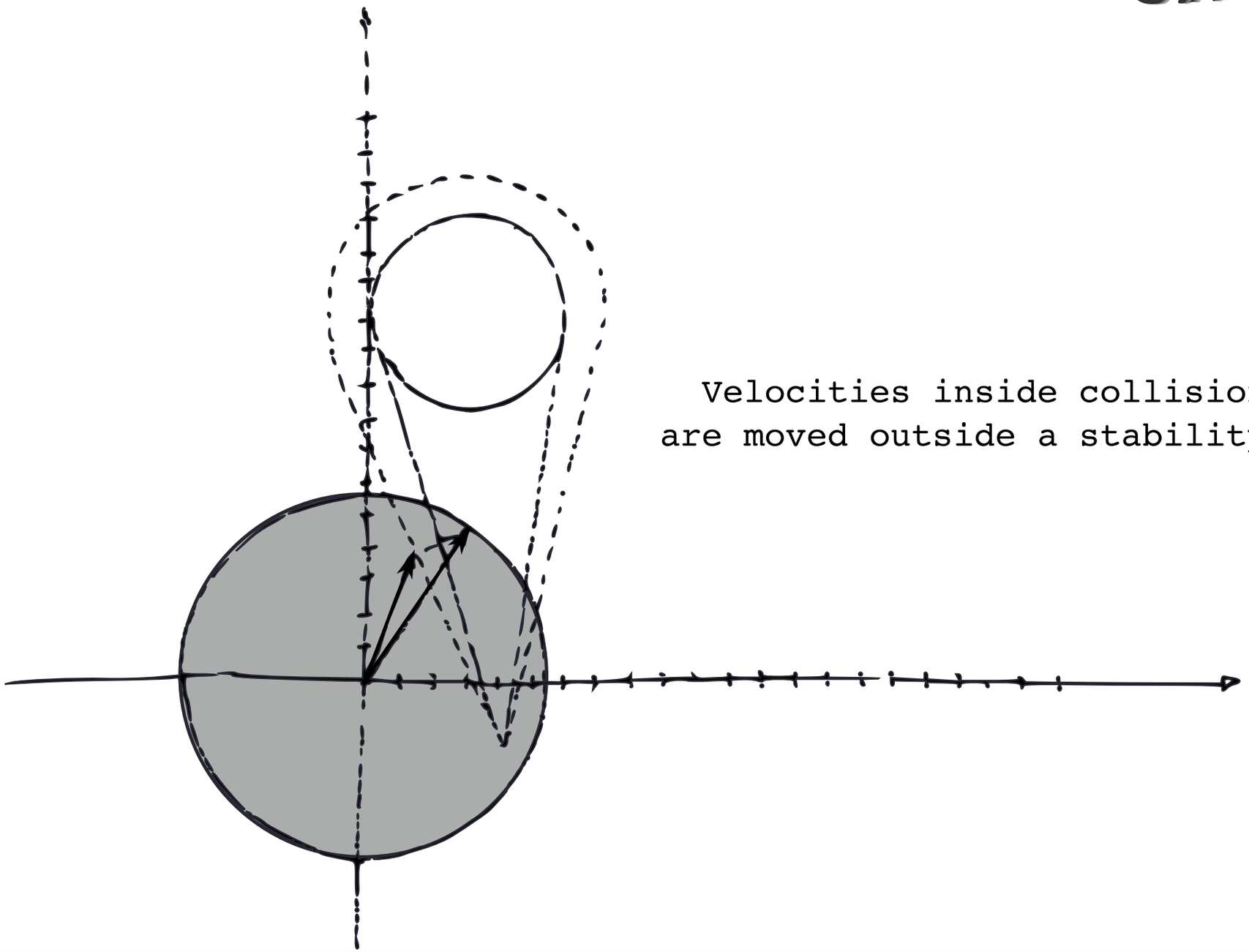


A unique velocity obstacle
is constructed for the obstacle

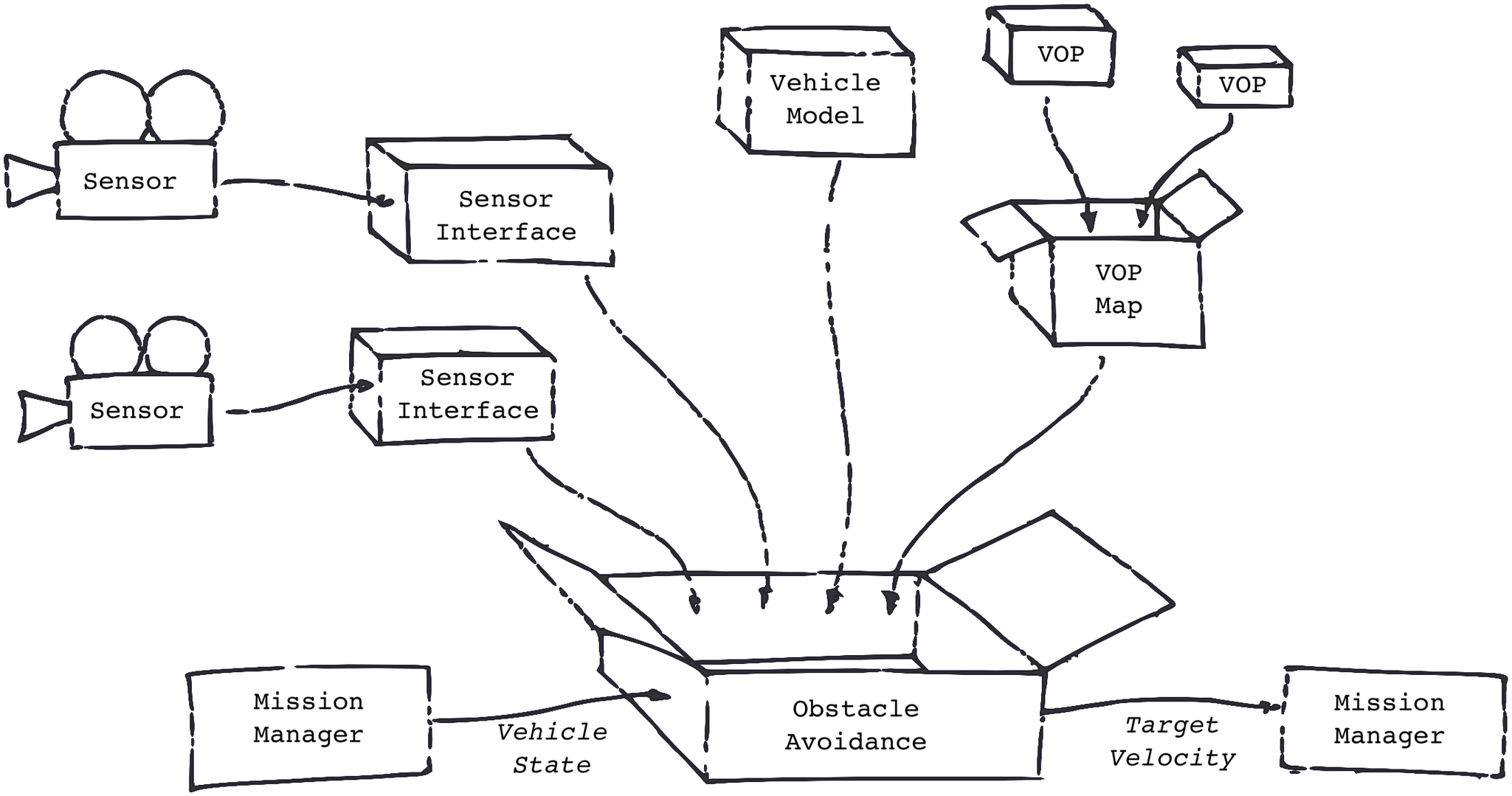


Velocity obstacle is adjusted based on the obstacle's velocity

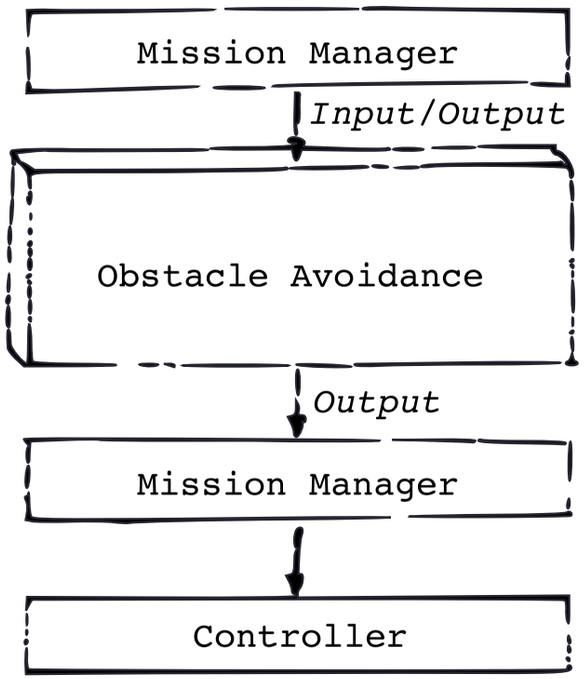

1-second collision time
Feasible velocity limit
Collision cut-off threshold
Infinite collision time



Velocities inside collision zone
are moved outside a stability buffer

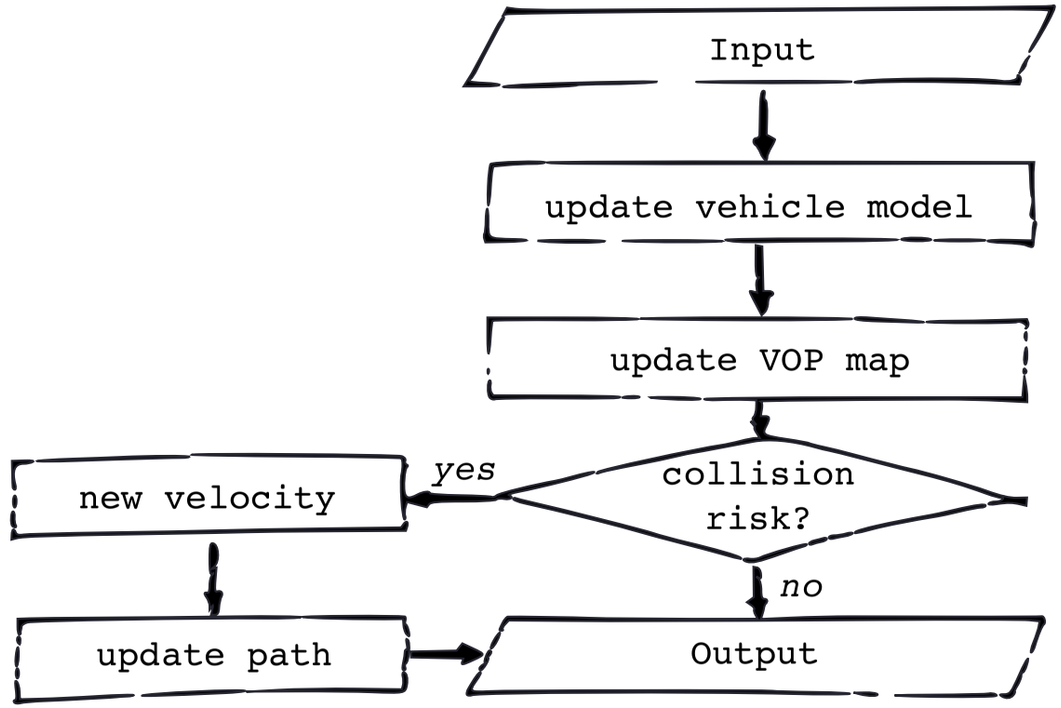

Sensor
Sensor Interface
Sensor
Sensor Interface
Vehicle Model
VOP
VOP
VOP Map
Mission Manager
Vehicle State
Obstacle Avoidance
Target Velocity
Mission Manager

```
#include "obstacleavoidance/ObstacleAvoidance.hpp"

...

bool CMissionManager::initMM( void )
{
  ...
  oa::ObstacleAvoidance::Instance()->init();
}

bool CMissionManager::runMM( void )
{
  ...
  supervisor.runSV();
  sequencecontroller.runSC();

  oa::ObstacleAvoidance::Instance
       ->run( input, trajectory, output );

  output.set_eAutoMission(
       condition.DetermineAutoMode() );
  ...
}

...
```



Process

```
void ObstacleAvoidance::run( Cinput, COuput, CTrajectory )
{
  if( !mObstacleAvoidanceActive )
    return;

  extractVehicleState();
  mVopMap.reset( mVehicle );

  for( SensorInterfaceIterator sensor = mSensors.begin();
       sensor = mSensors.end();
       sensor++ )
  {
    for( ObstacleIterator obs = (**sensor).begin();
         obs = (**sensor).end();
         obs++ )
    {
      obstacle->setReferencePosition(
                      mVehicle.getPosition() );
      mVopMap.update( *obstacle );
    }
  }

  ...

  if( timeToCollision < timeThreshold )
  {
     st_arCPath path = mVopMap.getClosestFeasibleVelocity(
                 extractTargetVelocity( output ) );
     output.set_arCPos( st_arCPos( CVector3D(0,0,0)) );
     output.set_arCPath( path );
     output.set_rCPsi( path.chi );
  }
}
```



- Additional Sensors
- Limited Discretization
- Third Dimension