# Dynamic Simulation of ASV Cluster Adaptively Navigating in a 2D Vector Field

Ryan Cooper
Garren Hendricks
Danop Rajabhandharaks

# Objective

Previous works have investigated how to find min/max point, to move in the direction of the field, to locate a ridge or trench.

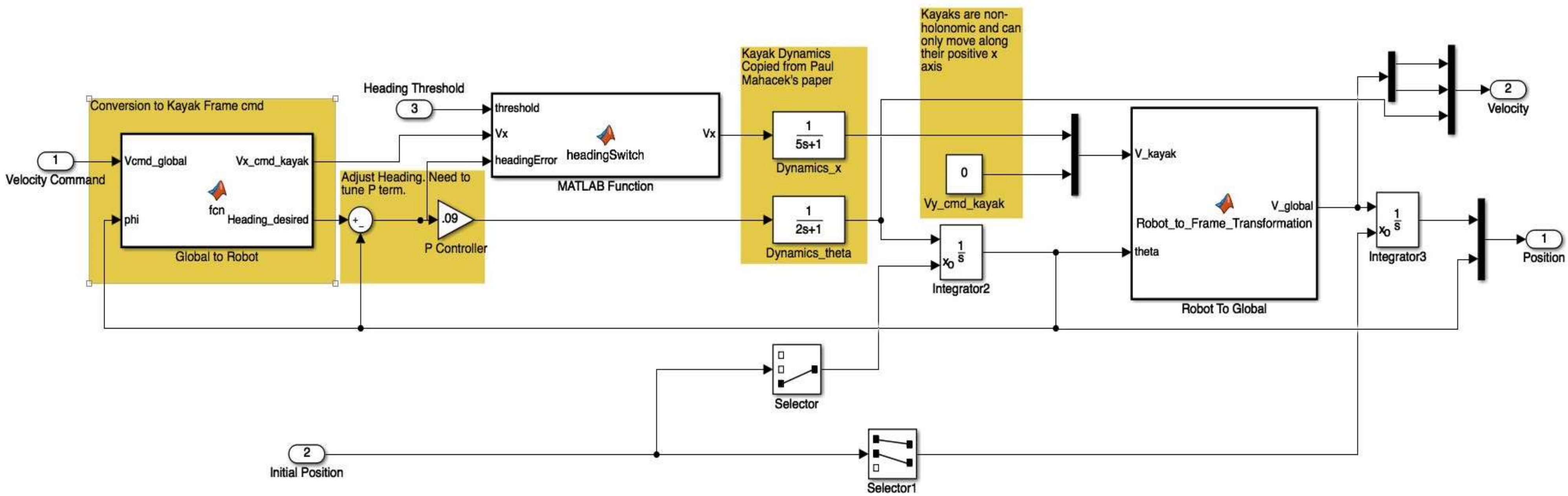
In this work, we expand our navigation strategy to include

- Contour Follower
- Trench Ascender

This model will be demonstrated by a real cluster of kayaks for adaptive navigation research.

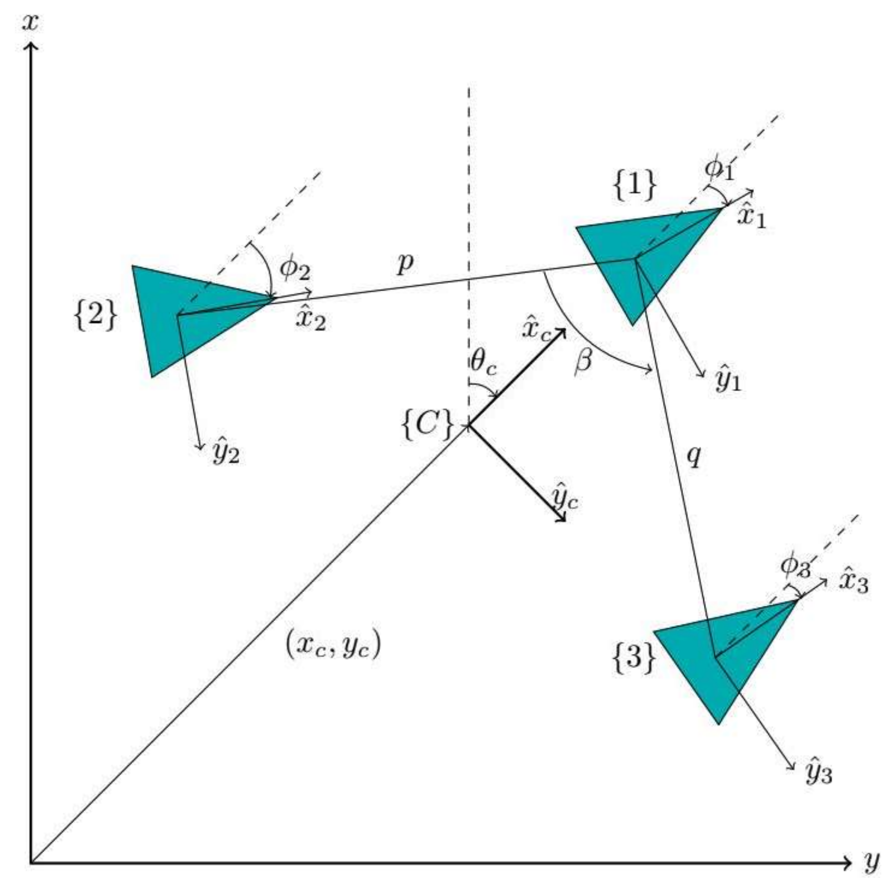
# Kayak Kinematics

Non holonomic model in simulink. Using velocity command to control.



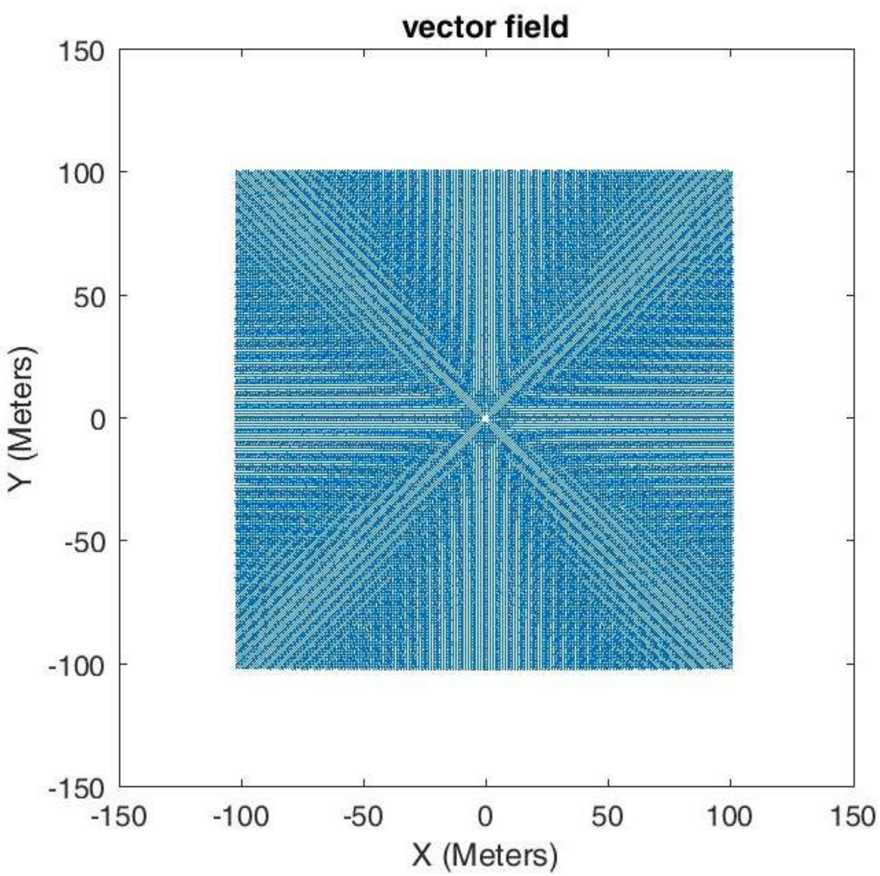
# Cluster Space Representation

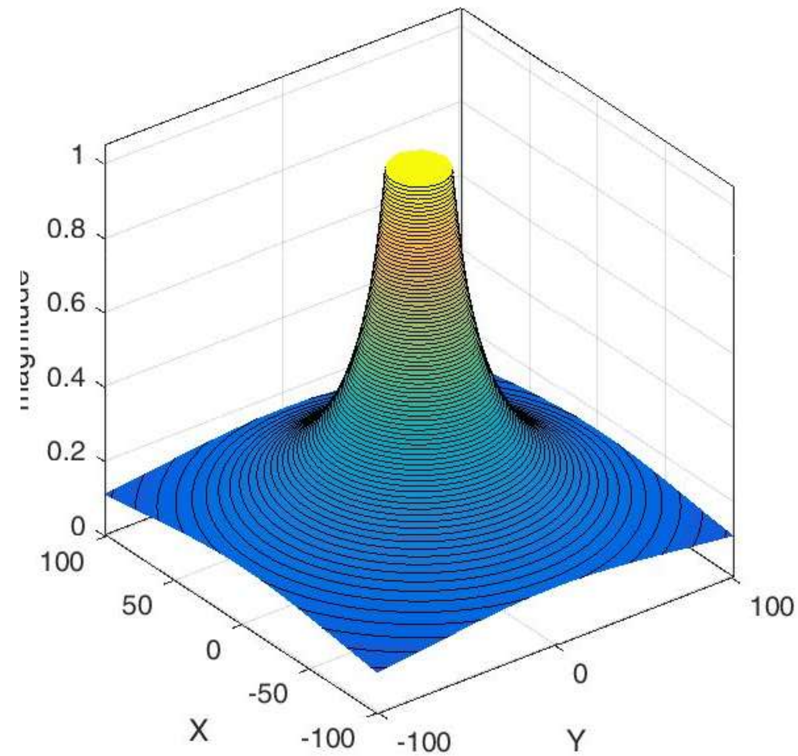
- Cluster is 3 robot centroid definition
- Cluster is holonomic, kayaks are not
- Kinematics and Jacobian switch when near beta singularity



# Inverse Jacobian

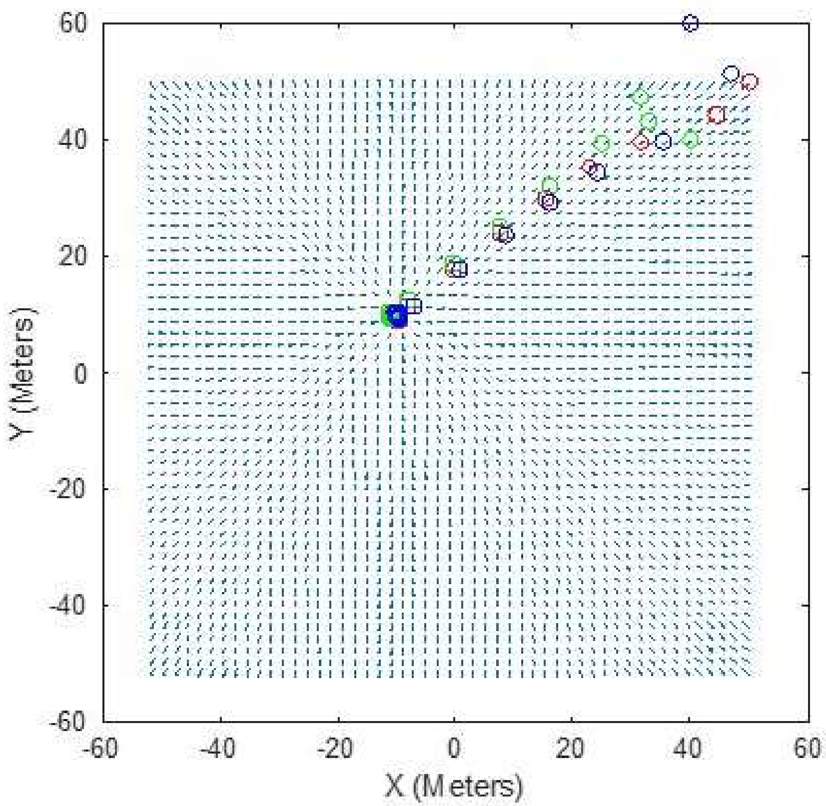
```
inv_Jac = [ 1, 0, -(sin(theta_c)*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2))/3, (cos(theta_c)*(2*p + 2*q*cos(beta1)))/(6*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)), (cos(theta_c)*(2*q + 2*p*cos(beta1)))/(6*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)),
-(p*q*sin(beta1)*cos(theta_c))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)), 0, 0, 0;

    0, 1,  (cos(theta_c)*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2))/3, (sin(theta_c)*(2*p + 2*q*cos(beta1)))/(6*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)), (sin(theta_c)*(2*q + 2*p*cos(beta1)))/(6*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)),
-(p*q*sin(beta1)*sin(theta_c))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)), 0, 0, 0;

    0, 0, 1, 0, 0, 0, 1, 0, 0;

    1, 0, p*cos(theta_c)*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2) + (sin(theta_c)*(2*p^2 + cos(beta1)*p*q - q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)),   sin(theta_c)*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q +
q^2))^(1/2) - (cos(theta_c)*(5*p*q^2 + 2*p^3 + 2*q^3*cos(beta1) + 6*p^2*q*cos(beta1) + p*q^2*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) + (p*q^2*sin(theta_c)*(p + q*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q +
q^2)^2*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), (cos(theta_c)*(4*p^2*q + q^3 + p^3*cos(beta1) + 3*p*q^2*cos(beta1) - p^2*q*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) - (p^2*q*sin(theta_c)*(p +
q*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), (p*q*sin(beta1)*cos(theta_c)*(- p^2 + cos(beta1)*p*q + 2*q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) +
(p*q^2*sin(beta1)*sin(theta_c)*(p^2*cos(beta1) + p*q*cos(beta1)^2 + p*q + q^2*cos(beta1)))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), 0, 0, 0;

    0, 1, p*sin(theta_c)*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2) - (cos(theta_c)*(2*p^2 + cos(beta1)*p*q - q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)), - cos(theta_c)*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q +
q^2))^(1/2) - (sin(theta_c)*(5*p*q^2 + 2*p^3 + 2*q^3*cos(beta1) + 6*p^2*q*cos(beta1) + p*q^2*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) - (p*q^2*cos(theta_c)*(p + q*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q +
q^2)^2*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), (sin(theta_c)*(4*p^2*q + q^3 + p^3*cos(beta1) + 3*p*q^2*cos(beta1) - p^2*q*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) + (p^2*q*cos(theta_c)*(p +
q*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), (p*q*sin(beta1)*sin(theta_c)*(- p^2 + cos(beta1)*p*q + 2*q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) -
(p*q^2*sin(beta1)*cos(theta_c)*(p^2*cos(beta1) + p*q*cos(beta1)^2 + p*q + q^2*cos(beta1)))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(q^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), 0, 0, 0;

    0, 0, 1, 0, 0, 0, 0, 1, 0;

    1, 0, (sin(theta_c)*(- p^2 + cos(beta1)*p*q + 2*q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)) - q*cos(theta_c)*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2),
(cos(theta_c)*(4*p*q^2 + p^3 + q^3*cos(beta1) + 3*p^2*q*cos(beta1) - p*q^2*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) + (p*q^2*sin(theta_c)*(q + p*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q +
q^2)^2*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), - sin(theta_c)*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2) - (cos(theta_c)*(5*p^2*q + 2*q^3 + 2*p^3*cos(beta1) + 6*p*q^2*cos(beta1) +
p^2*q*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) - (p^2*q*sin(theta_c)*(q + p*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)),
(p*q*sin(beta1)*cos(theta_c)*(2*p^2 + cos(beta1)*p*q - q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) - (p^2*q*sin(beta1)*sin(theta_c)*(p^2*cos(beta1) + p*q*cos(beta1)^2 + p*q + q^2*cos(beta1)))/((p^2 + 2*cos(beta1)*p*q +
q^2)^2*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), 0, 0, 0;

    0, 1, - q*sin(theta_c)*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2) - (cos(theta_c)*(- p^2 + cos(beta1)*p*q + 2*q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(1/2)), (sin(theta_c)*(4*p*q^2 + p^3 + q^3*cos(beta1) +
3*p^2*q*cos(beta1) - p*q^2*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) - (p*q^2*cos(theta_c)*(q + p*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)),
cos(theta_c)*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2) - (sin(theta_c)*(5*p^2*q + 2*q^3 + 2*p^3*cos(beta1) + 6*p*q^2*cos(beta1) + p^2*q*cos(beta1)^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) + (p^2*q*cos(theta_c)*(q +
p*cos(beta1))*(cos(beta1)^2 - 1))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)),   (p*q*sin(beta1)*sin(theta_c)*(2*p^2 + cos(beta1)*p*q - q^2))/(3*(p^2 + 2*cos(beta1)*p*q + q^2)^(3/2)) +
(p^2*q*sin(beta1)*cos(theta_c)*(p^2*cos(beta1) + p*q*cos(beta1)^2 + p*q + q^2*cos(beta1)))/((p^2 + 2*cos(beta1)*p*q + q^2)^2*(-(p^2*(cos(beta1)^2 - 1))/(p^2 + 2*cos(beta1)*p*q + q^2))^(1/2)), 0, 0, 0;

    0, 0, 1, 0, 0, 0, 0, 0, 1];
```


vector field
150
100
50
0
-50
-100
-150
-150
-100
-50
0
50
100
150
X (Meters)
Y (Meters)


1
0.8
0.6
0.4
0.2
0
100
50
0
-50
-100
-100
0
100
X
Y

# Max/Min Response Finder Algorithm

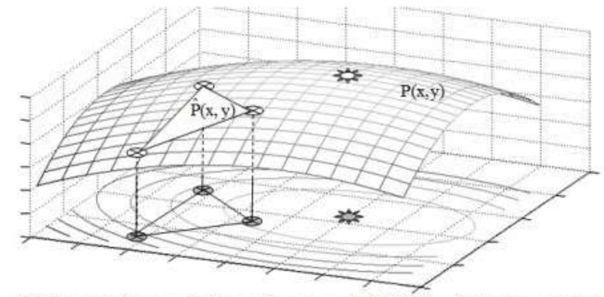
- Follow the gradient



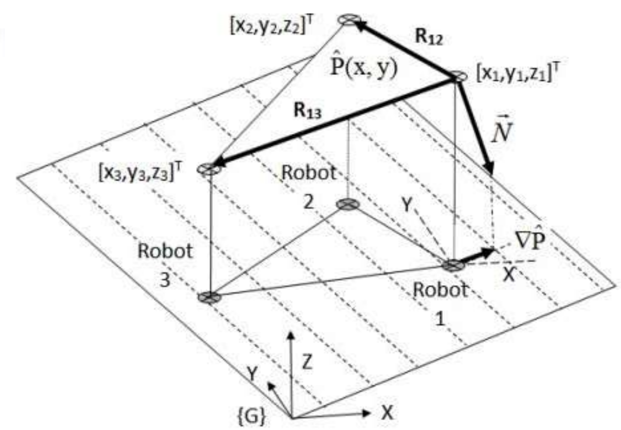

$$\vec{N} = -\begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix} \times \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{pmatrix}$$

$$\nabla \hat{P} = [N_x, N_y]^T$$

$$b_{grad} = \mathrm{ATAN2}(N_y, N_x)$$



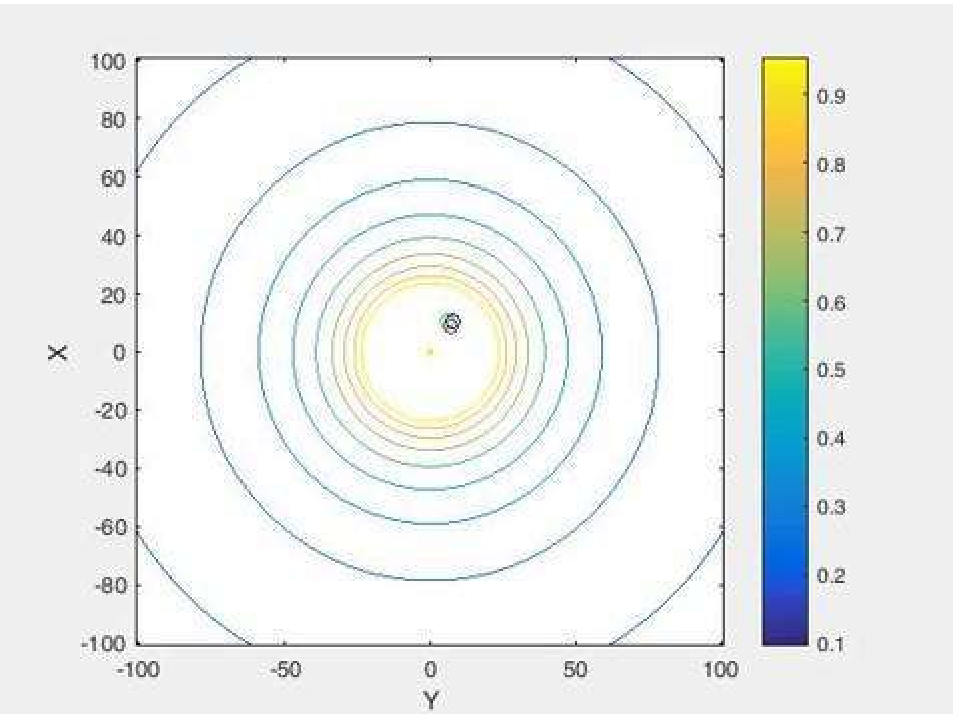

(a) Three robots sample the scalar parameter field P(x,y), thereby creating a local approximation in the form of the plane $\hat{P}(x,y)$.

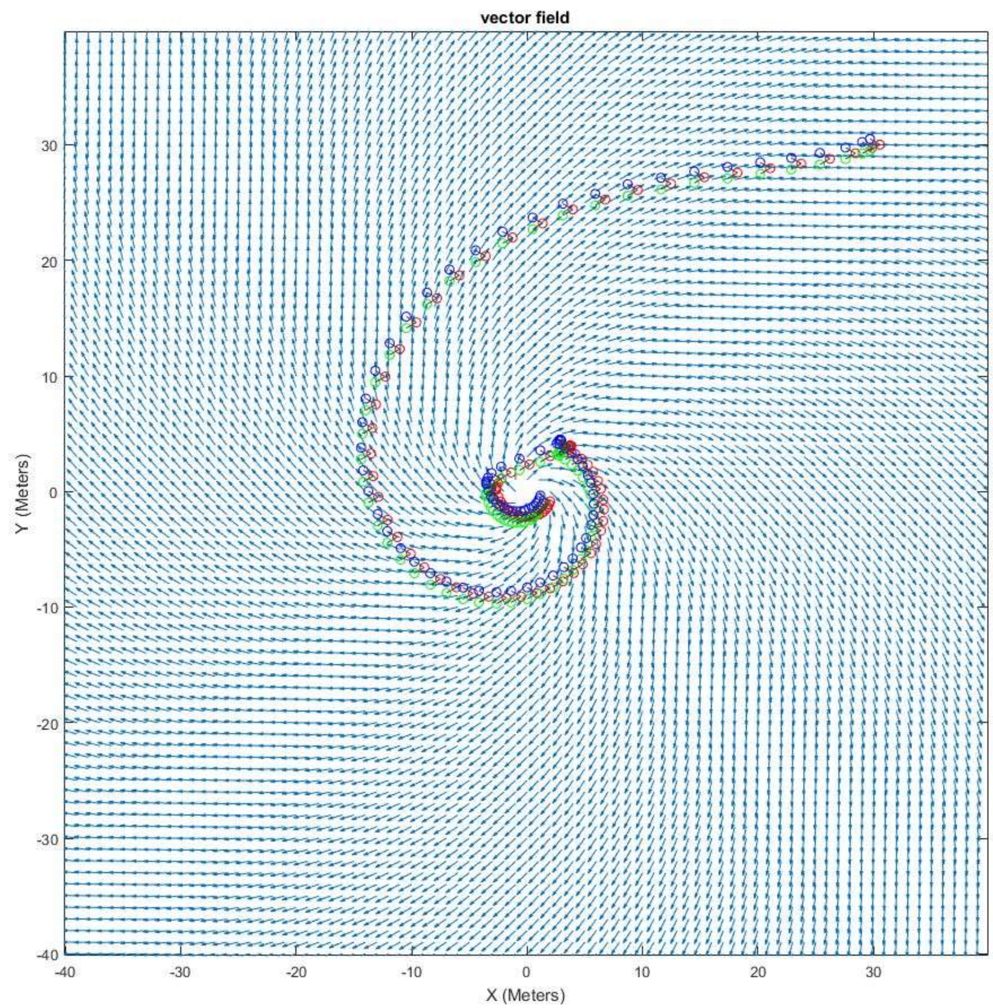



(b) The three robots define vectors within the planar field approximation, allowing the direction of the field gradient to be computed.

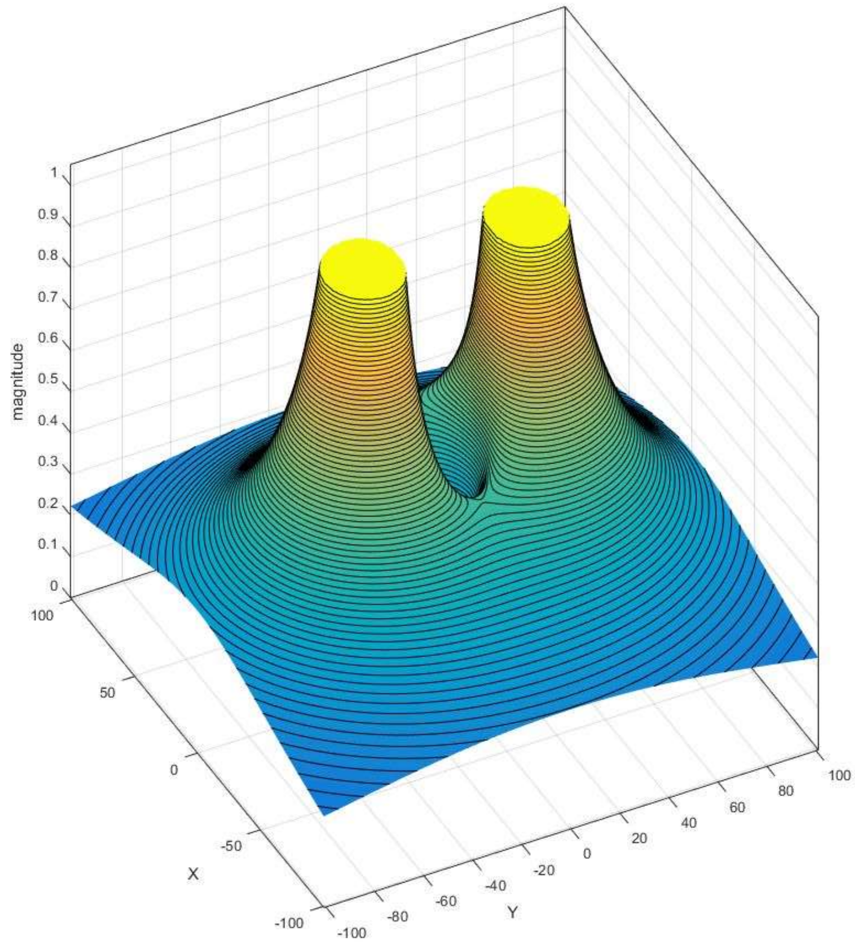
# Direction Follower

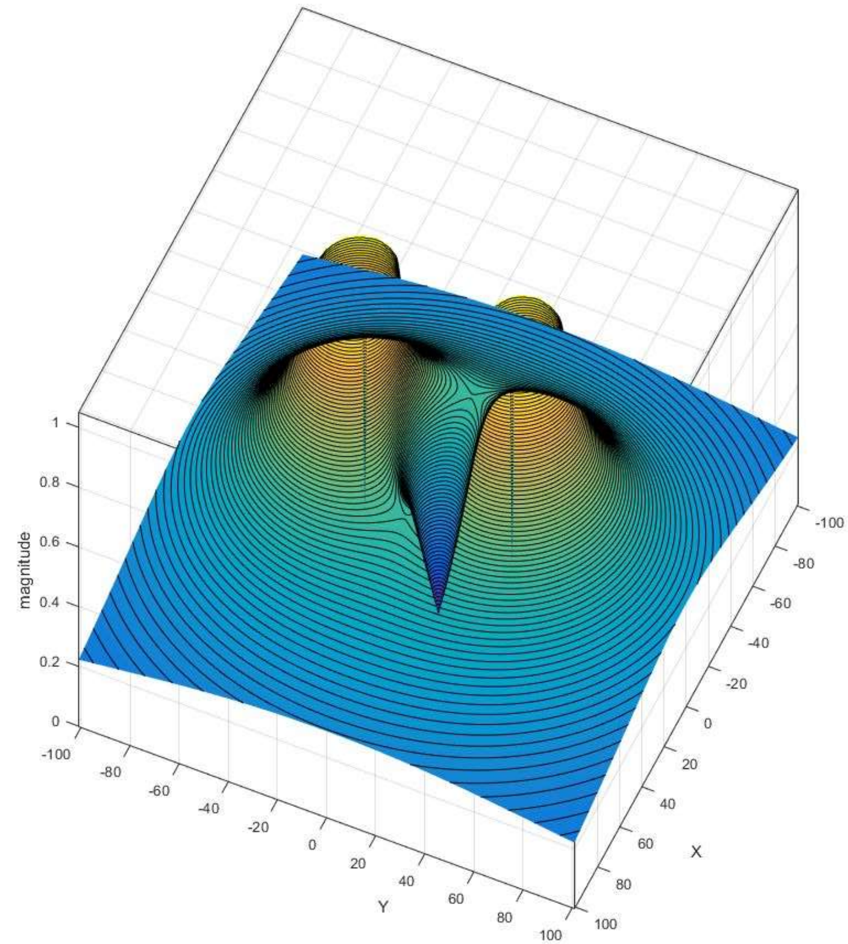
- Go in the direction of the vector field



# Double Source

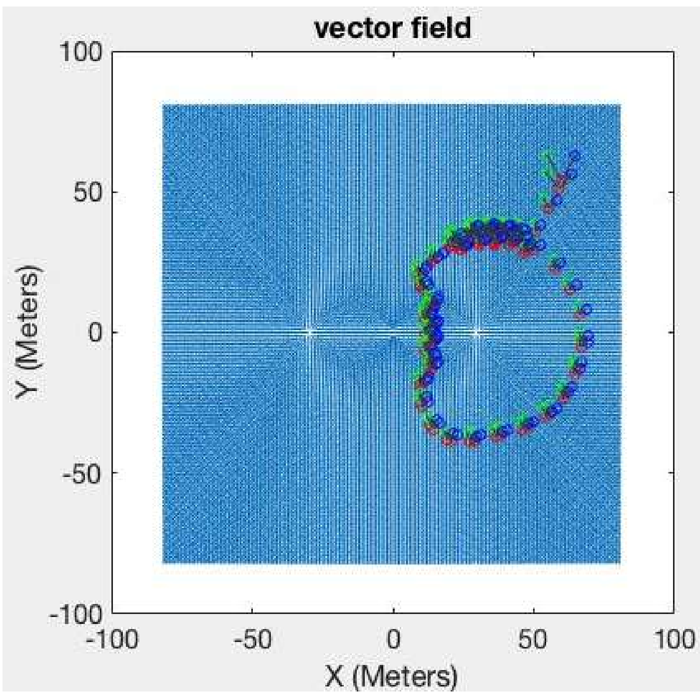


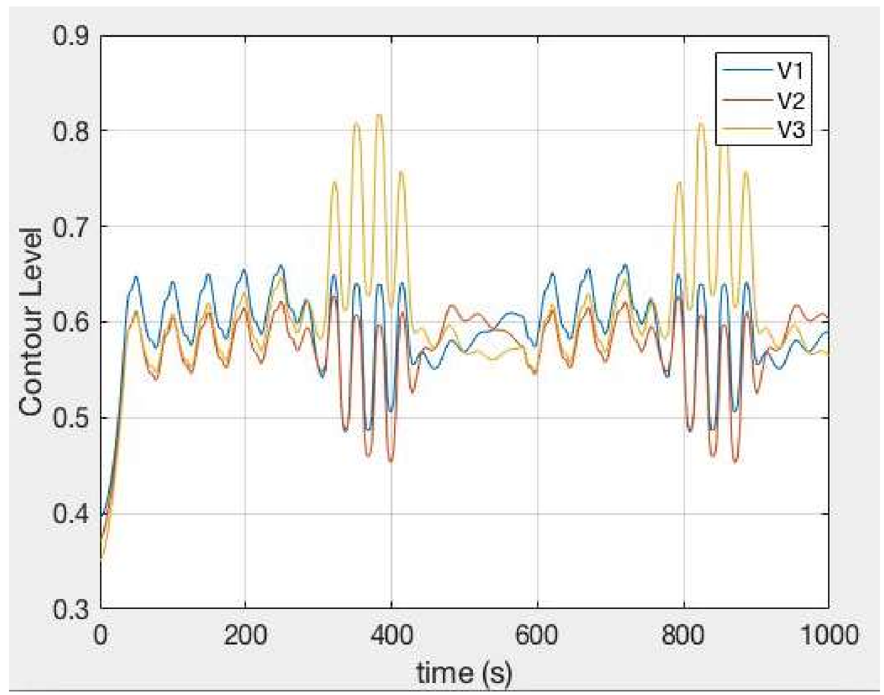



# Contour Follower Algorithm

$$\begin{aligned} b_{cf} &= b_{grad} \\ &+d\{sgn(z_{des} - z_c) \times min[K_{ct} \times \|z_{des} - z_c\|, \pi/2] - (\pi/2)\} \end{aligned}$$

Zdes = 0.6
Shape Variable p = q = 3
Kct = 20

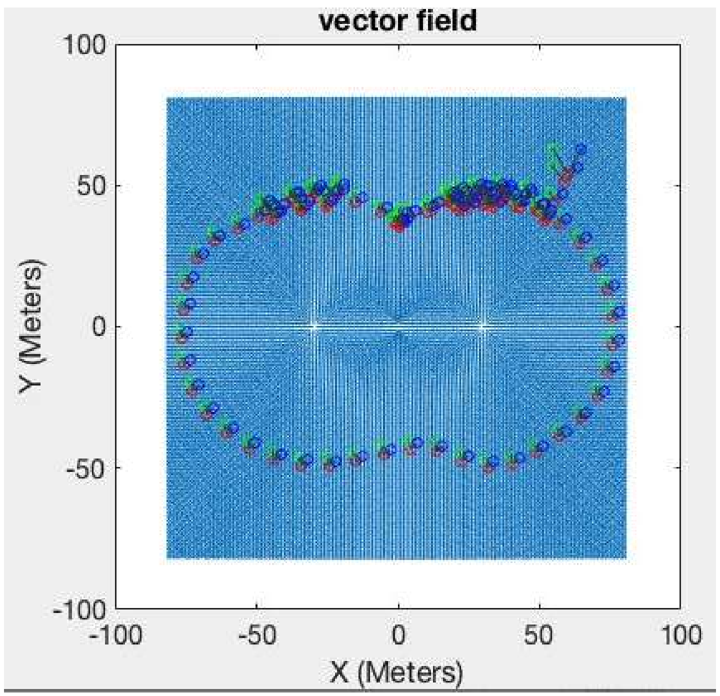


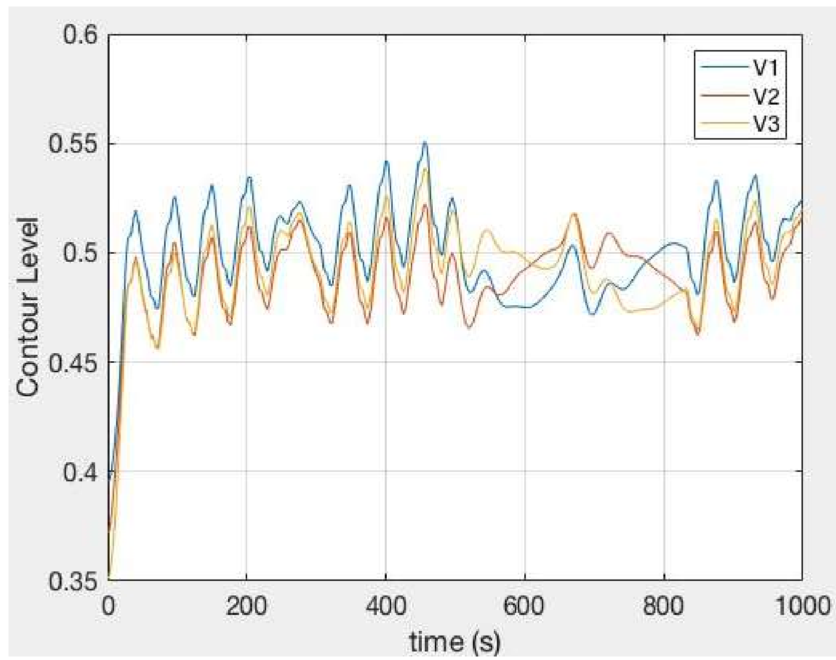



# Contour Follower Algorithm

$$b_{cf} = b_{grad} + d\{sgn(z_{des} - z_c) \times min[K_{ct} \times \|z_{des} - z_c\|, \pi/2] - (\pi/2)\}$$

Zdes = 0.5
Shape Variable p = q = 3
Kct = 20

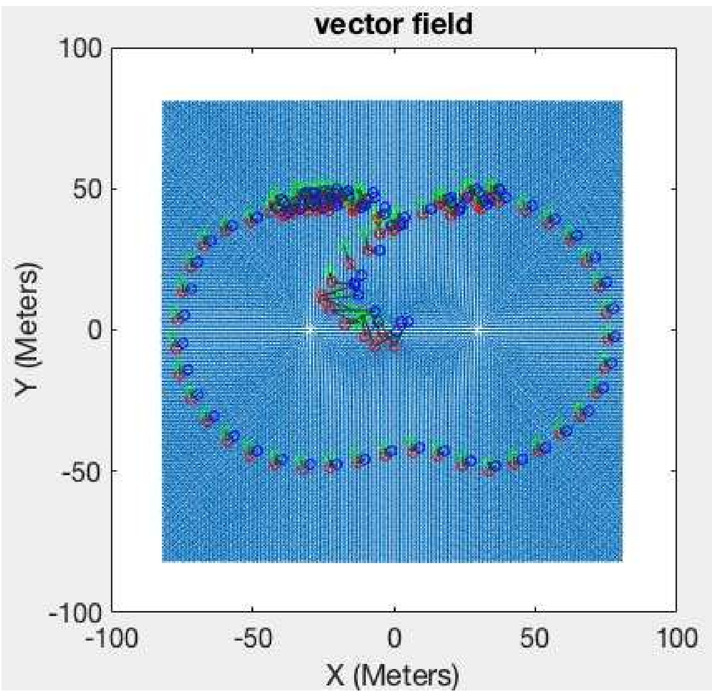


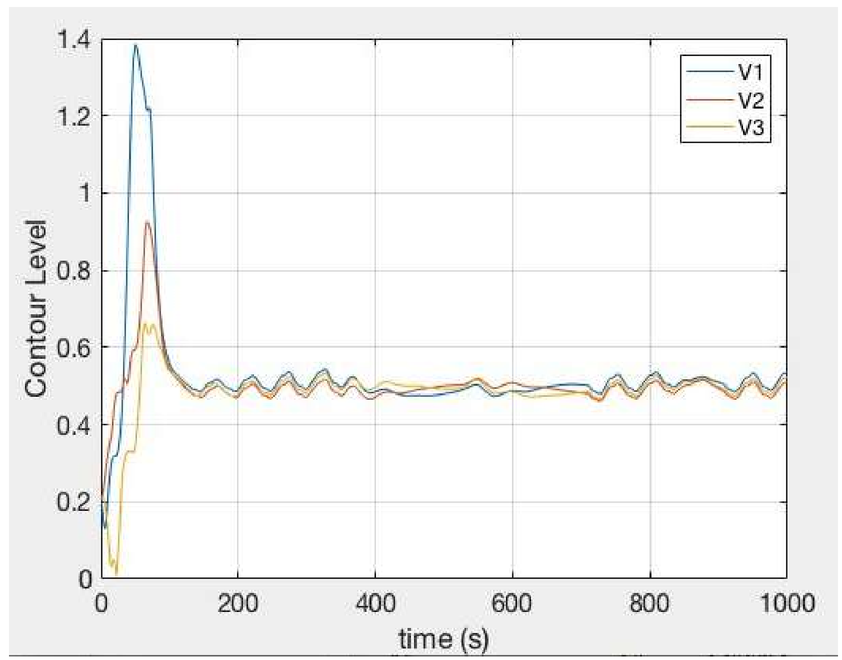



# Contour Follower Algorithm

$$b_{cf} = b_{grad} + d\{sgn(z_{des} - z_c) \times min[K_{ct} \times \|z_{des} - z_c\|, \pi/2] - (\pi/2)\}$$

Zdes = 0.5
Shape Variable p = q = 3
Kct = 20

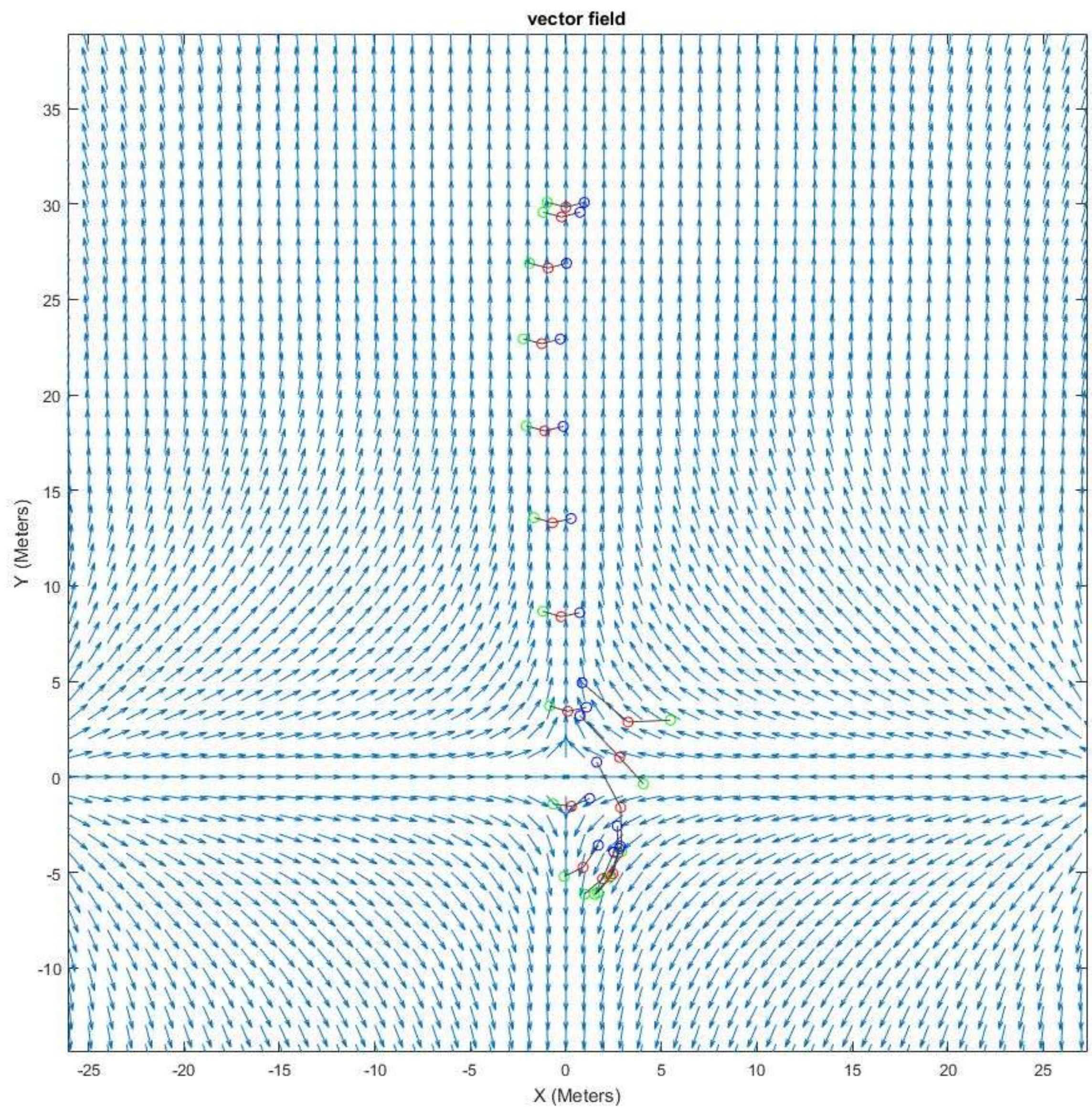


[redacted]

# Trench Ascent

- Starts on top of trench
- 3 control efforts
  a. Stay on trench
  b. Stay perpendicular to trench
  c. Move up the gradient



# Conclusion

- Simulation of kayak dynamics
- Replication of previous particle simulation work
- Contour following
- Trench ascension