BACKGROUND
GRAPH-BASED FORMATION

- Using the equations from [1]
  - $e_{i,j}$ is the measured range
  - $b_{i,j}$ is the desired range
  - $\alpha_{i,j}$ is the measured bearing

- Infrared sensors to get range/bearing

\[ \begin{align*}
\delta_{x,j} &= \sum_{j=1}^{\Delta} [- \mathcal{L}_{i,j} \cdot (e_{i,j} - b_{i,j}) \cdot \cos(\alpha_{i,j})] \\
\delta_{y,j} &= \sum_{j=1}^{\Delta} [- \mathcal{L}_{i,j} \cdot (e_{i,j} - b_{i,j}) \cdot \sin(\alpha_{i,j})] \\
\delta_i(e_{i,j}(t), \alpha_i(t)) &= \sqrt{\delta_{x,i}^2(t) + \delta_{y,i}^2(t)} \\
\bar{\alpha}_i(e_{i,j}(t), \alpha_i(t)) &= \arctan(\delta_{y,i}(t), \delta_{x,i}(t)) \\
u_i(t) &= K_u \cdot \delta_i(e_{i,j}(t), \alpha_i(t)) \cdot \cos(\bar{\alpha}_i(e_{i,j}(t), \alpha_i(t))) \\
\omega_i(t) &= K_\omega \cdot \bar{\alpha}_i(e_{i,j}(t), \alpha_i(t))
\end{align*} \]

WEBOTS
WEBOTS IMPLEMENTATION

- Straightforward
- Using wb_… functions to send/receive messages
- Send/receive is fast and reliable

Algorithm:
- Receive until the buffer is empty
- Send once
- Compute the new speeds
- Do it again
/*receive information */
while (wb_receiver_get_queue_length(receiver) > 0 && count < FLOCK_SIZE) {
    inbuffer = (char*) wb_receiver_get_data(receiver);
    sscanf(inbuffer, "%d", &rob_nb);  //get the robot number
    range[rob_nb] = (float)(sqrt(1.0/wb_receiver_get_signal_strength(receiver))) - position_range[robot_id][rob_nb];  //get the robot range
    bearing[rob_nb] = compute_bearing(wb_receiver_get_emitter_direction(receiver));
    count++;
    wb_receiver_next_packet(receiver);
}

/*send information */
sprintf(outbuffer, "%1d", robot_id);
wtf_emitter_send(emitter, outbuffer, strlen(outbuffer));

/*Compute speeds*/
compute_speeds(range, bearing, &u, &w);
compute_wheel_speeds(u, w, &msl, &msr);
wtf_differential_wheels_set_speed(msl, msr);
WEBOTS IMPLEMENTATION
REAL E-PUCKS
REAL E-PUCK IMPLEMENTATION

- Obstacles to overcome:
  - Noise
  - Interference from other e-pucks
  - Incorrect values
  - Data acquisition frequency

- Using LibIRcom library to communicate

- Algorithm:
  - Every 250ms: update the speeds
  - Receive as long as the buffer is not empty, for at least 600ms
  - Send one message
  - Loop
e_activate_agenda(formation, 2500); //Computing the speeds
last_time = ircomGetTime0;
while(1)
{
    while((ircomIsReceiving0 == 1) || (ircomGetTime0 < last_time + 6000))
    {
        receive_ir0 == 1;
    }
    last_time = ircomGetTime0;
    send_ir0;
}
REAL E-PUCK IMPLEMENTATION

Filtering in receive function:

```c
int value = (int) imsg.value;
if(value < 4 && value != robot_id)
{
    tmp_range = ((float)imsg.distance)-position_range[robot_id][value]; // get the robot range
    if (fabs(tmp_range-range[value]) < 5.0 ) range[value] = tmp_range;
    else range[value] += (tmp_range-range[value])/fabs(tmp_range-range[value])*5.0;
    bearing[value] = (float)imsg.direction;
    if(bearing[value] > PI) bearing[value] -= (2.0*PI);
    if (range[value] < 1 && range[value] > -1) range[value] = 0; //Threshold to be in position
}
```
REAL E-PUCK IMPLEMENTATION
PERFORMANCE
PERFORMANCE COMPARISON

- The fitness values range from 0 to 1
- Higher performance leads to higher values in fitness
- Fitness depends on the range and bearing of the e-pucks
  - $d_{\text{diff}}$ is between leader and each follower
  - $\alpha_{\text{diff}}$ is between a pair of followers, as seen by the leader

\[
d_{\text{diff}} = \frac{\text{abs}(\text{actual\_range} - \text{desired\_range})}{\text{desired\_range}} \quad \alpha_{\text{diff}} = \frac{\text{abs}(\text{actual\_angle} - \text{desired\_angle})}{\text{desired\_angle}}
\]

\[
\text{fitness} = e^{-\frac{\Sigma d_{\text{diff}}^2 + \Sigma \alpha_{\text{diff}}}{\text{FLOCK\_SIZE} - 1}}
\]
PERFORMANCE COMPARISON

```cpp
for (i = 1; i < FLOCK_SIZE; i++) {
    fitness += fabs(range[i])/position_range[0][i];
    for (j = i+1; j < FLOCK_SIZE; j++) {
        float rel_angle = fabs(bearing[i]-bearing[j]);
        if (rel_angle > PI) rel_angle = 2 * PI - rel_angle;
        float dist0i2 = position_range[0][i] * position_range[0][i];
        float dist0j2 = position_range[0][j] * position_range[0][j];
        float distij2 = position_range[i][j] * position_range[i][j];
        float wanted_angle = acos((dist0i2 + dist0j2 - distij2)/(2 * position_range[0][i] * position_range[0][j]));
        float difference = fabs(wanted_angle - rel_angle);
        fitness += difference/wanted_angle;
    }
}
fitness = exp(-fitness/(float)(FLOCK_SIZE-1));
```
PERFORMANCE COMPARISON

![Graph showing performance comparison between two types of robots (epuck and webots). The graph plots fitness over time.](image)
CONCLUSION
Conclusion

- Webots is a good starting point: easy to test, fast changes
- But perfect case
- Need to take time for real e-pucks
- Infrared is noisy
- Trial and error
- Communication rate is important
- Simple algorithm can achieve good formation
QUESTIONS?

Thank you for your attention!