KIT
Karlsruhe Institute of Technology

RF 2.0 RESEARCH FACILITY

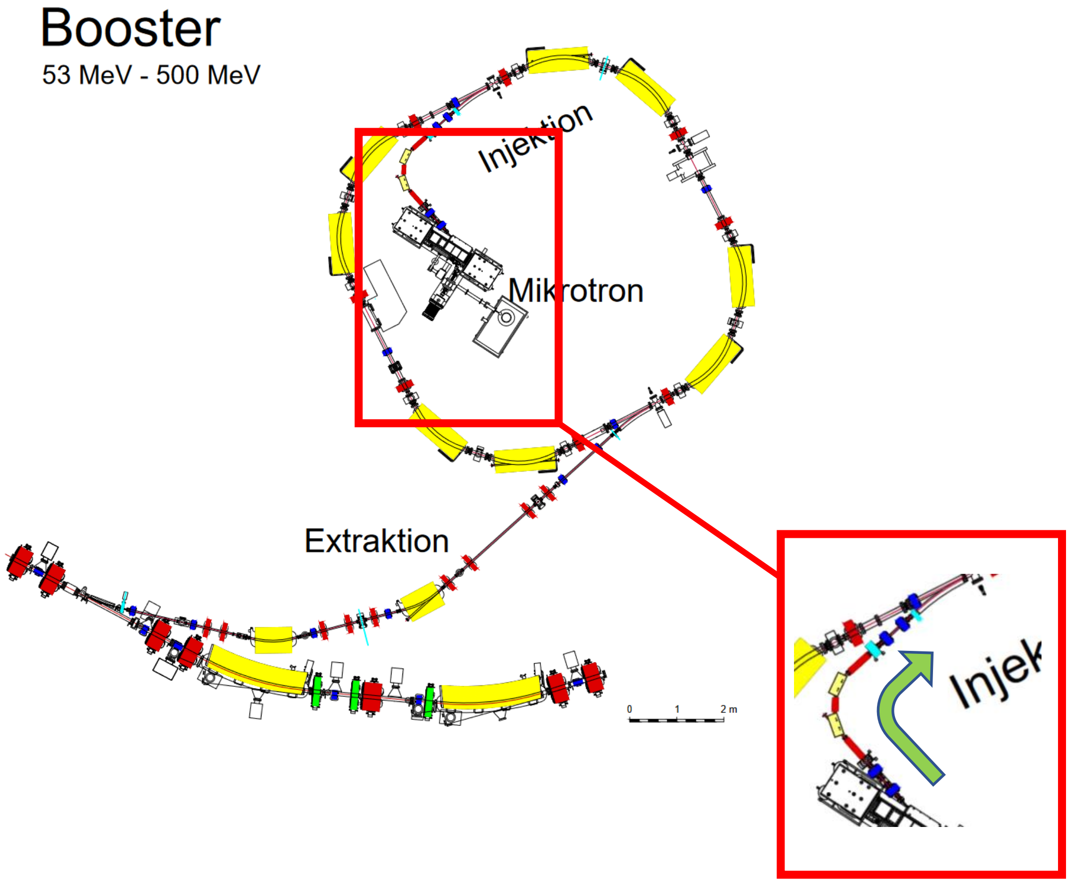
Efficient Accelerator Operation with Artificial Intelligence Based Optimization Methods

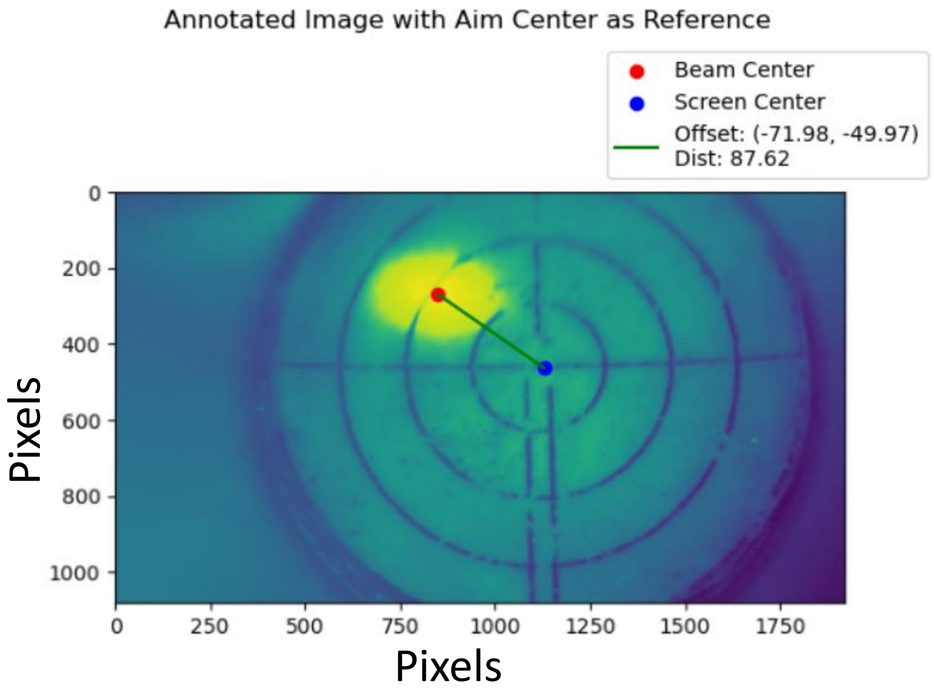
Evangelos Matzoukas

Real-Time Systems for Energy Technologies Group @ Institute for Technical Physics (ITEP)

CASE STUDY: Misalignment of the injection line beam of KARA Accelerator

Karlsruhe Institute of Technology





Main causes :
hysteresis, **calibration errors**, or **deviations from the modeled lattice**.

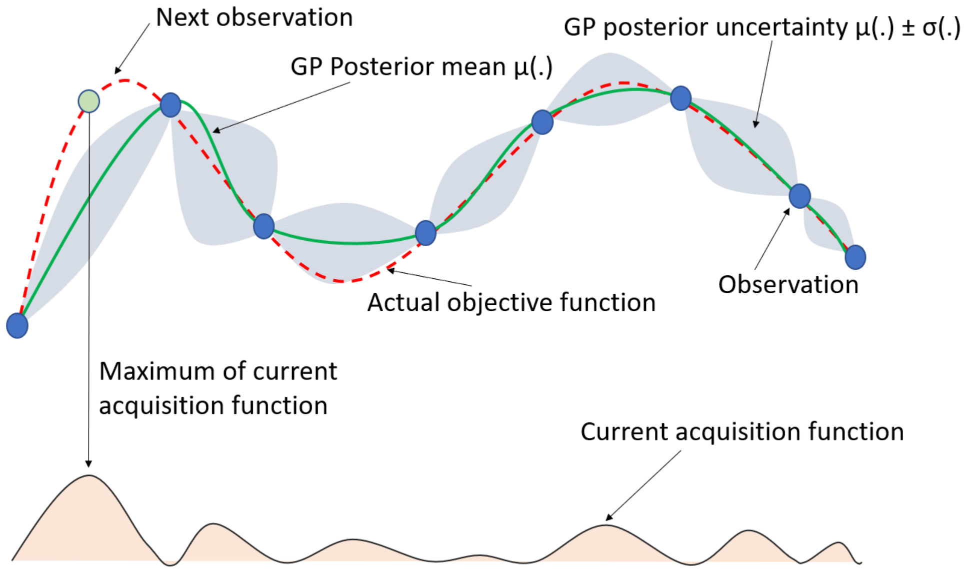
- **Deflection of the particle beam** in the injection line as it passes through the quadrupole magnets.
- **Increased Energy Loss**.
- **Reduced Beam Efficiency and Performance**.
- **Unreliable Experimental and Image Data** leading to errors in diagnostics.

CASE STUDY: Misalignment of the injection line beam of KARA Accelerator

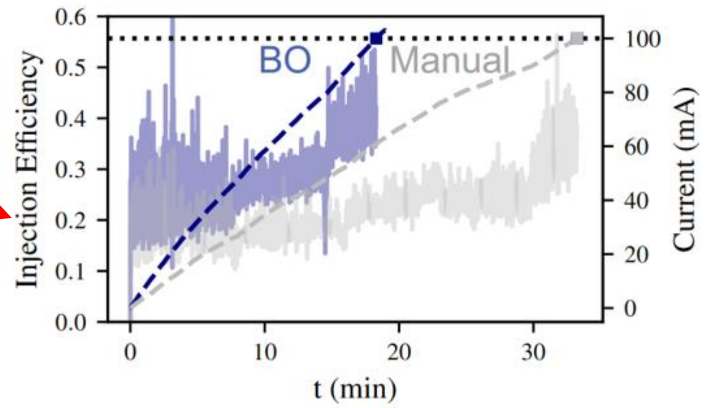
Karlsruhe Institute of Technology

Solution : Application of Bayesian Optimization (BO) to control steering and quadrupole magnets in KARA

- **Faster tuning** with fewer tests
- **Accurate alignment** by handling complex parameter interactions.
- **Automated optimization** reduces manual effort.



Optimizations with comparable machine condition.
Reduced injection time for 100 mA from 30 min to 18 min.

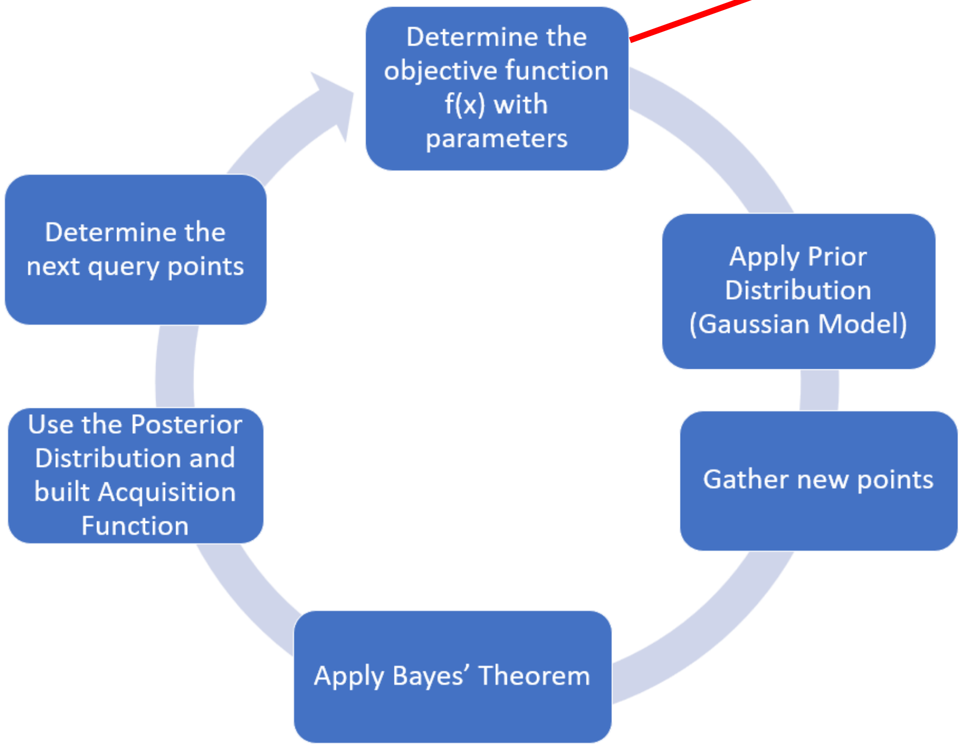


(C.Xu et al. 2023)

Bayesian Optimization in Accelerator Tuning

Karlsruhe Institute of Technology

How does it work?



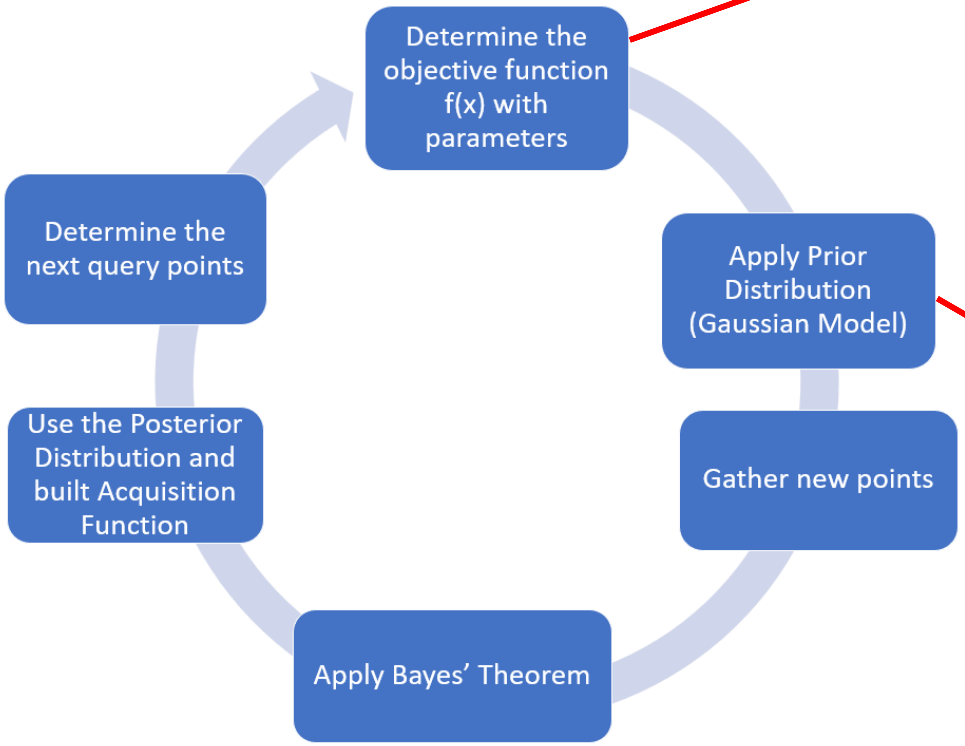

```python
coordinates = []
angles_recorded = []

# Define the objective function for Bayesian Optimization
def fun(angles):
    global coordinates

    segment = get_segment()
    # Set angles
    segment.M1.angle = torch.tensor(angles["angle_1"])
    segment.M2.angle = torch.tensor(angles["angle_2"])

    # Track beam
    outgoing_beam = segment.track(incoming_beam)

    # Get sensor reading
    array = segment.SCREEN.reading.numpy()

    # Get the coordinates of the max value
    coords = np.unravel_index(np.argmax(array), array.shape)
    coordinates.append(coords)

    # Store angles
    angles_recorded.append((angles["angle_1"], angles["angle_2"]))

    # Compute Manhattan distance from center
    return {"distance": abs(coords[0] - 1024/2) + abs(coords[1] - 1024/2)}

# Define the Bayesian optimization problem
vocs = VOCS(
    variables={
        "angle_1": [-6e-4, 6e-4],  # Search range for angle 1
        "angle_2": [-6e-4, 6e-4],  # Search range for angle 2
    },
    objectives={"distance": "MINIMIZE"},  # Minimize distance to center
)
```

Define obj. function with varying angles to control 2 steering magnets

Bayesian Optimization in Accelerator Tuning

Karlsruhe Institute of Technology

How does it work?




```python
coordinates = []
angles_recorded = []

# Define the objective function for Bayesian Optimization
def fun(angles):
    global coordinates

    segment = get_segment()
    # Set angles
    segment.M1.angle = torch.tensor(angles["angle_1"])
    segment.M2.angle = torch.tensor(angles["angle_2"])

    # Track beam
    outgoing_beam = segment.track(incoming_beam)

    # Get sensor reading
    array = segment.SCREEN.reading.numpy()

    # Get the coordinates of the max value
    coords = np.unravel_index(np.argmax(array), array.shape)
    coordinates.append(coords)

    # Store angles
    angles_recorded.append((angles["angle_1"], angles["angle_2"]))

    # Compute Manhattan distance from center
    return {"distance": abs(coords[0] - 1024/2) + abs(coords[1] - 1024/2)}

# Define the Bayesian optimization problem
vocs = VOCS(
    variables={
        "angle_1": [-6e-4, 6e-4],  # Search range for angle 1
        "angle_2": [-6e-4, 6e-4],  # Search range for angle 2
    },
    objectives={"distance": "MINIMIZE"},  # Minimize distance to center
)

# Create Bayesian Optimization Generator
generator = UpperConfidenceBoundGenerator(vocs=vocs)

# Define the evaluator
evaluator = Evaluator(function=fun)

# Set up Xopt with Bayesian Optimization
xopt = Xopt(generator=generator, evaluator=evaluator, vocs=vocs)

# Run optimization
xopt.random_evaluate(20)
#xopt.run()  # Runs for 20 evaluations

for i in range(50):
    xopt.step()

# Get results
display("Best Parameters:", xopt.data.nsmallest(1, "distance"))
```

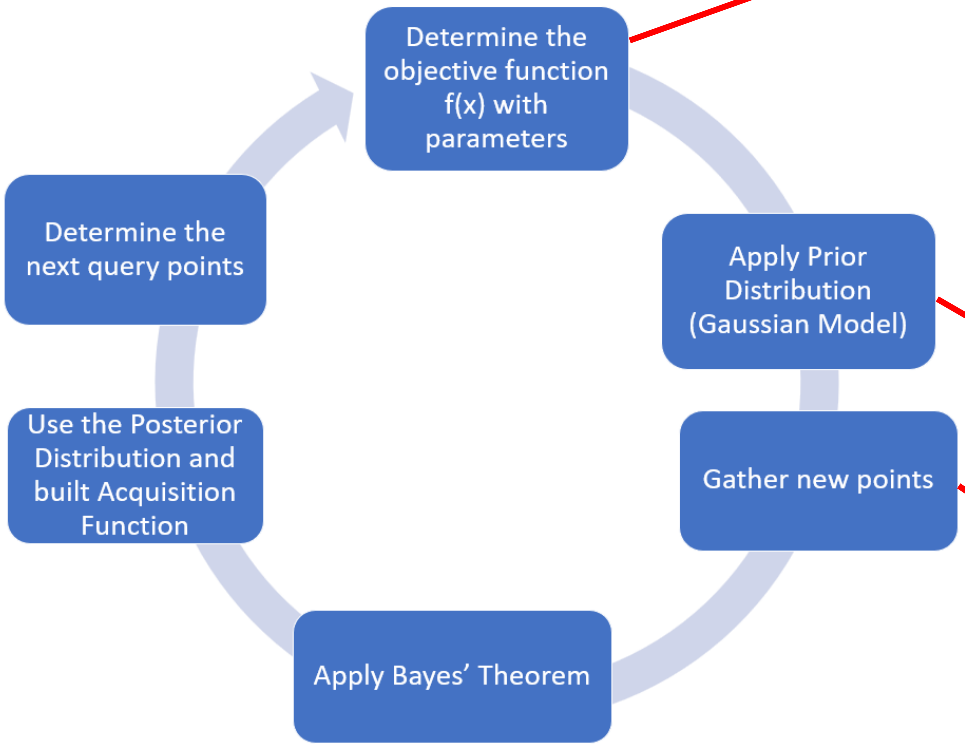
Define obj. function with varying angles to control 2 steering magnets

This sets up the Gaussian model

Bayesian Optimization in Accelerator Tuning

Karlsruhe Institute of Technology

How does it work?



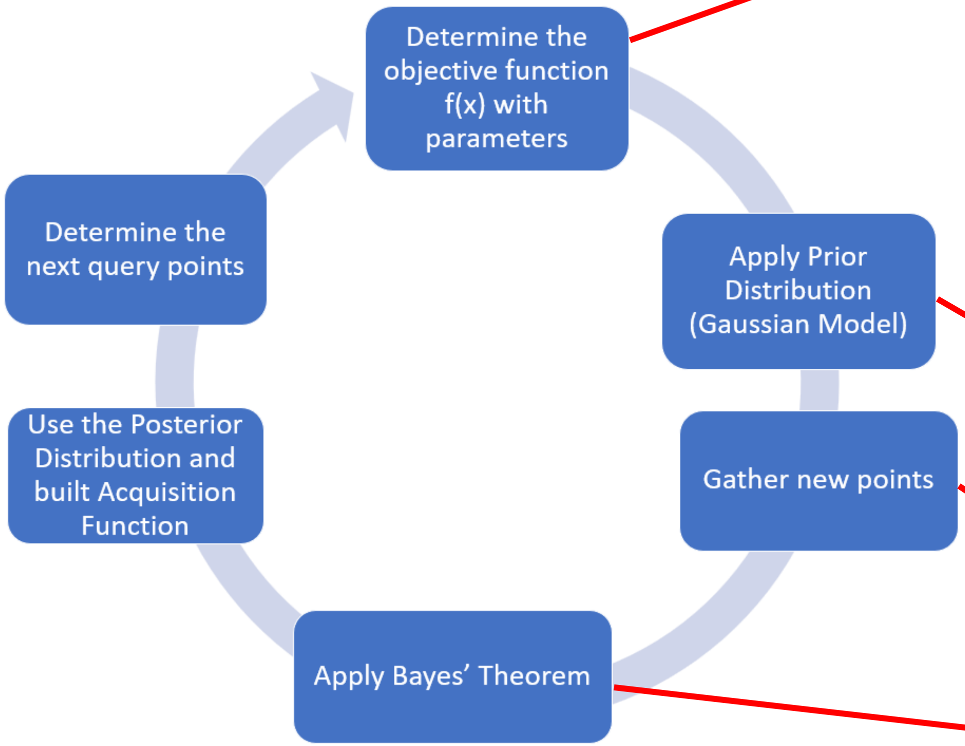

```
coordinates = []
angles_recorded = []

# Define the objective function for Bayesian Optimization
def fun(angles):
    global coordinates

    segment = get_segment()
    # Set angles
    segment.M1.angle = torch.tensor(angles["angle_1"])
    segment.M2.angle = torch.tensor(angles["angle_2"])

    # Track beam
    outgoing_beam = segment.track(incoming_beam)

    # Get sensor reading
    array = segment.SCREEN.reading.numpy()

    # Get the coordinates of the max value
    coords = np.unravel_index(np.argmax(array), array.shape)
    coordinates.append(coords)

    # Store angles
    angles_recorded.append((angles["angle_1"], angles["angle_2"]))

    # Compute Manhattan distance from center
    return {"distance": abs(coords[0] - 1024/2) + abs(coords[1] - 1024/2)}

# Define the Bayesian optimization problem
vocs = VOCS(
    variables={
        "angle_1": [-6e-4, 6e-4],  # Search range for angle 1
        "angle_2": [-6e-4, 6e-4],  # Search range for angle 2
    },
    objectives={"distance": "MINIMIZE"},  # Minimize distance to center
)

# Create Bayesian Optimization Generator
generator = UpperConfidenceBoundGenerator(vocs=vocs)

# Define the evaluator
evaluator = Evaluator(function=fun)

# Set up Xopt with Bayesian Optimization
xopt = Xopt(generator=generator, evaluator=evaluator, vocs=vocs)

# Run optimization
xopt.random_evaluate(20)
#xopt.run()  # Runs for 20 evaluations

for i in range(50):
    xopt.step()

# Get results
display("Best Parameters:", xopt.data.nsmallest(1, "distance"))
```

Define obj. function with varying angles to control 2 steering magnets

This sets up the Gaussian model

Bayesian Optimization in Accelerator Tuning

How does it work?




```
coordinates = []
angles_recorded = []

# Define the objective function for Bayesian Optimization
def fun(angles):
    global coordinates

    segment = get_segment()
    # Set angles
    segment.M1.angle = torch.tensor(angles["angle_1"])
    segment.M2.angle = torch.tensor(angles["angle_2"])

    # Track beam
    outgoing_beam = segment.track(incoming_beam)

    # Get sensor reading
    array = segment.SCREEN.reading.numpy()

    # Get the coordinates of the max value
    coords = np.unravel_index(np.argmax(array), array.shape)
    coordinates.append(coords)

    # Store angles
    angles_recorded.append((angles["angle_1"], angles["angle_2"]))

    # Compute Manhattan distance from center
    return {"distance": abs(coords[0] - 1024/2) + abs(coords[1] - 1024/2)}

# Define the Bayesian optimization problem
vocs = VOCS(
    variables={
        "angle_1": [-6e-4, 6e-4],  # Search range for angle 1
        "angle_2": [-6e-4, 6e-4],  # Search range for angle 2
    },
    objectives={"distance": "MINIMIZE"},  # Minimize distance to center
)

# Create Bayesian Optimization Generator
generator = UpperConfidenceBoundGenerator(vocs=vocs)

# Define the evaluator
evaluator = Evaluator(function=fun)

# Set up Xopt with Bayesian Optimization
xopt = Xopt(generator=generator, evaluator=evaluator, vocs=vocs)

# Run optimization
xopt.random_evaluate(20)
#xopt.run()  # Runs for 20 evaluations

for i in range(50):
    xopt.step()

# Get results
display("Best Parameters:", xopt.data.nsmallest(1, "distance"))
```

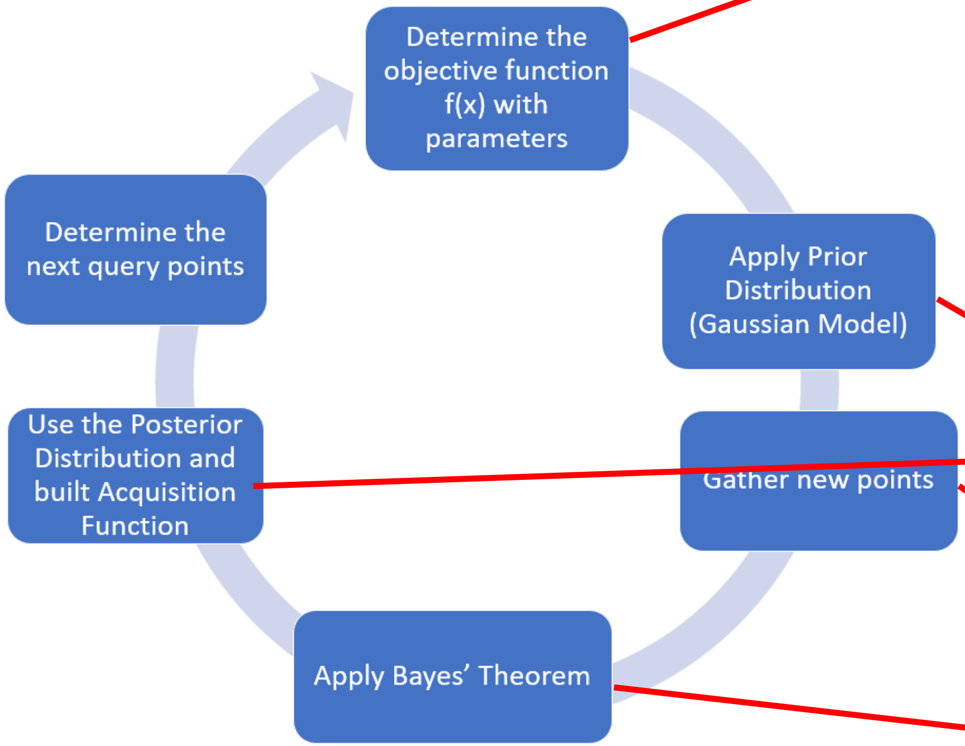
Define obj. function with varying angles to control 2 steering magnets

This sets up the Gaussian model

Updates Gaussian model and applies Bayes Theorem

Bayesian Optimization in Accelerator Tuning

How does it work?




```python
coordinates = []
angles_recorded = []

# Define the objective function for Bayesian Optimization
def fun(angles):
    global coordinates

    segment = get_segment()
    # Set angles
    segment.M1.angle = torch.tensor(angles["angle_1"])
    segment.M2.angle = torch.tensor(angles["angle_2"])

    # Track beam
    outgoing_beam = segment.track(incoming_beam)

    # Get sensor reading
    array = segment.SCREEN.reading.numpy()

    # Get the coordinates of the max value
    coords = np.unravel_index(np.argmax(array), array.shape)
    coordinates.append(coords)

    # Store angles
    angles_recorded.append((angles["angle_1"], angles["angle_2"]))

    # Compute Manhattan distance from center
    return {"distance": abs(coords[0] - 1024/2) + abs(coords[1] - 1024/2)}

# Define the Bayesian optimization problem
vocs = VOCS(
    variables={
        "angle_1": [-6e-4, 6e-4],  # Search range for angle 1
        "angle_2": [-6e-4, 6e-4],  # Search range for angle 2
    },
    objectives={"distance": "MINIMIZE"},  # Minimize distance to center
)
# Create Bayesian Optimization Generator
generator = UpperConfidenceBoundGenerator(vocs=vocs)

# Define the evaluator
evaluator = Evaluator(function=fun)

# Set up Xopt with Bayesian Optimization
xopt = Xopt(generator=generator, evaluator=evaluator, vocs=vocs)

# Run optimization
xopt.random_evaluate(20)
#xopt.run()  # Runs for 20 evaluations

for i in range(50):
    xopt.step()

# Get results
display("Best Parameters:", xopt.data.nsmallest(1, "distance"))
```

Define obj. function with varying angles to control 2 steering magnets

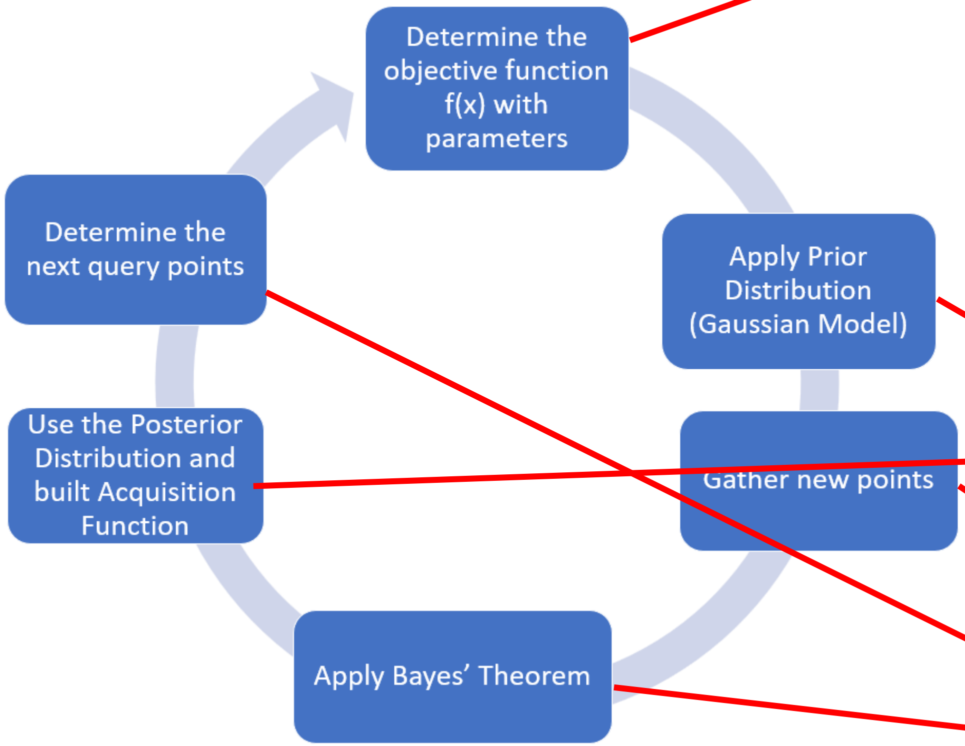
This sets up the Gaussian model

Updates Gaussian model and applies Bayes Theorem

Bayesian Optimization in Accelerator Tuning

Karlsruhe Institute of Technology

How does it work?




```python
coordinates = []
angles_recorded = []

# Define the objective function for Bayesian Optimization
def fun(angles):
    global coordinates

    segment = get_segment()
    # Set angles
    segment.M1.angle = torch.tensor(angles["angle_1"])
    segment.M2.angle = torch.tensor(angles["angle_2"])

    # Track beam
    outgoing_beam = segment.track(incoming_beam)

    # Get sensor reading
    array = segment.SCREEN.reading.numpy()

    # Get the coordinates of the max value
    coords = np.unravel_index(np.argmax(array), array.shape)
    coordinates.append(coords)

    # Store angles
    angles_recorded.append((angles["angle_1"], angles["angle_2"]))

    # Compute Manhattan distance from center
    return {"distance": abs(coords[0] - 1024/2) + abs(coords[1] - 1024/2)}

# Define the Bayesian optimization problem
vocs = VOCS(
    variables={
        "angle_1": [-6e-4, 6e-4],  # Search range for angle 1
        "angle_2": [-6e-4, 6e-4],  # Search range for angle 2
    },
    objectives={"distance": "MINIMIZE"},  # Minimize distance to center
)

# Create Bayesian Optimization Generator
generator = UpperConfidenceBoundGenerator(vocs=vocs)

# Define the evaluator
evaluator = Evaluator(function=fun)

# Set up Xopt with Bayesian Optimization
xopt = Xopt(generator=generator, evaluator=evaluator, vocs=vocs)

# Run optimization
xopt.random_evaluate(20)
#xopt.run()  # Runs for 20 evaluations

for i in range(50):
    xopt.step()

# Get results
display("Best Parameters:", xopt.data.nsmallest(1, "distance"))
```

Define obj. function with varying angles to control 2 steering magnets

This sets up the Gaussian model

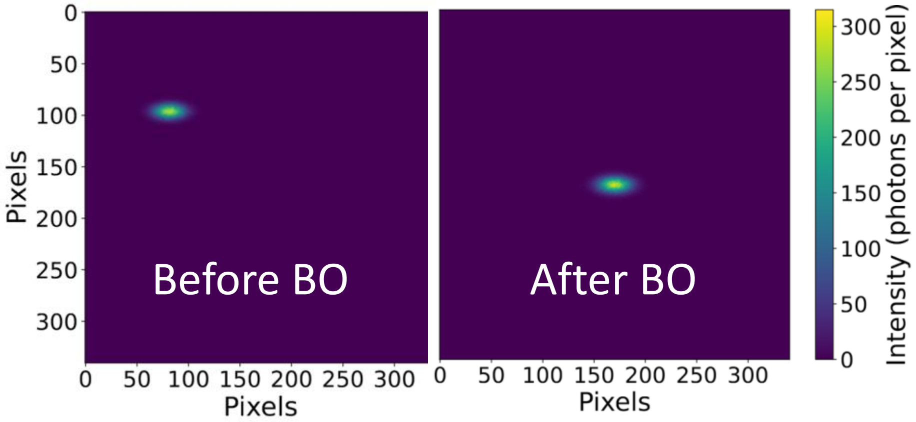
Updates Gaussian model and applies Bayes Theorem

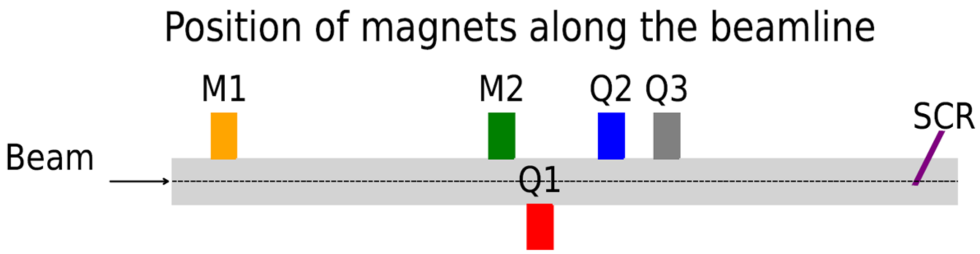
Repeatedly **choosing the next best point to evaluate**, using the acquisition function Exploitation / Exploration

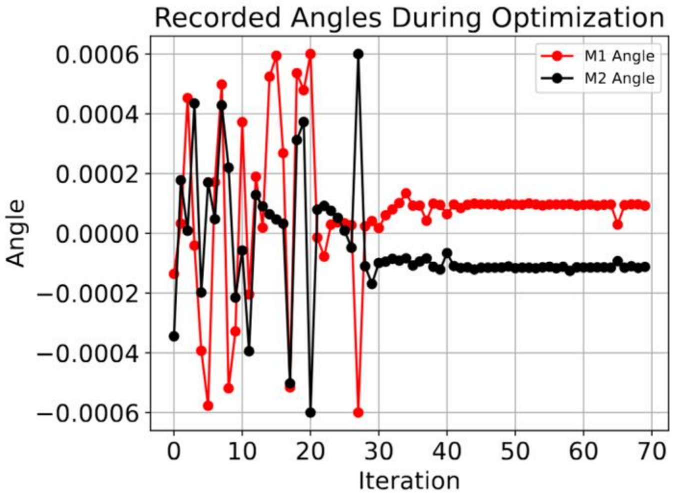
Results

Control of the steering magnets

Cheetah Simulation software: https://github.com/desy-ml/cheetah





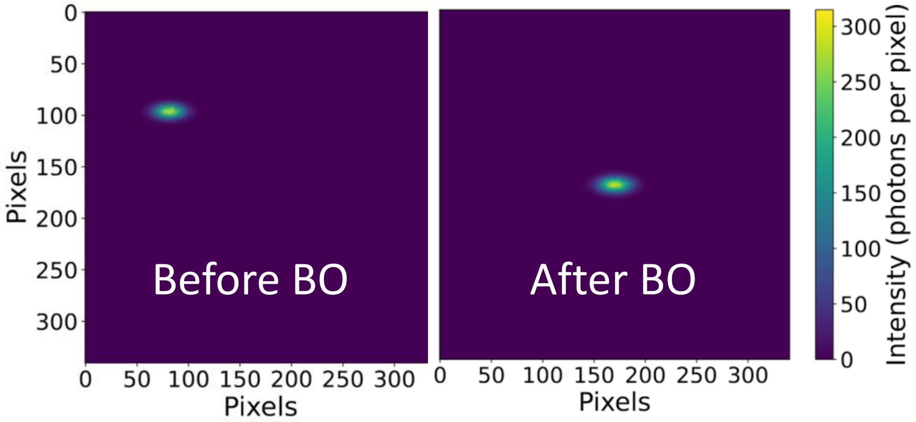


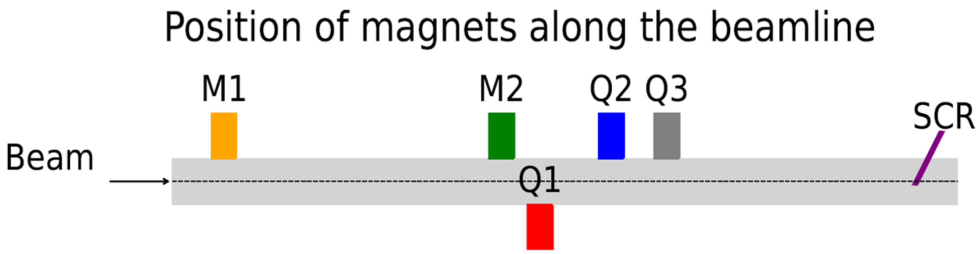
- Successful centering of the beam in the centre of the screen after (BO)

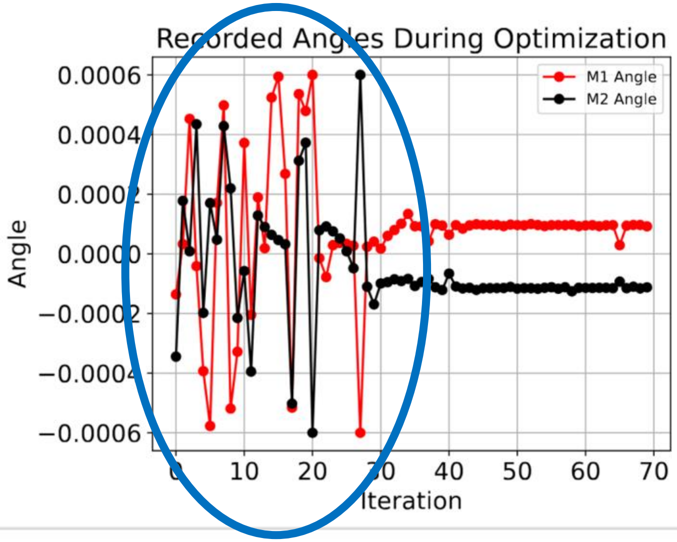
Results

Control of the steering magnets

Cheetah Simulation software: https://github.com/desy-ml/cheetah



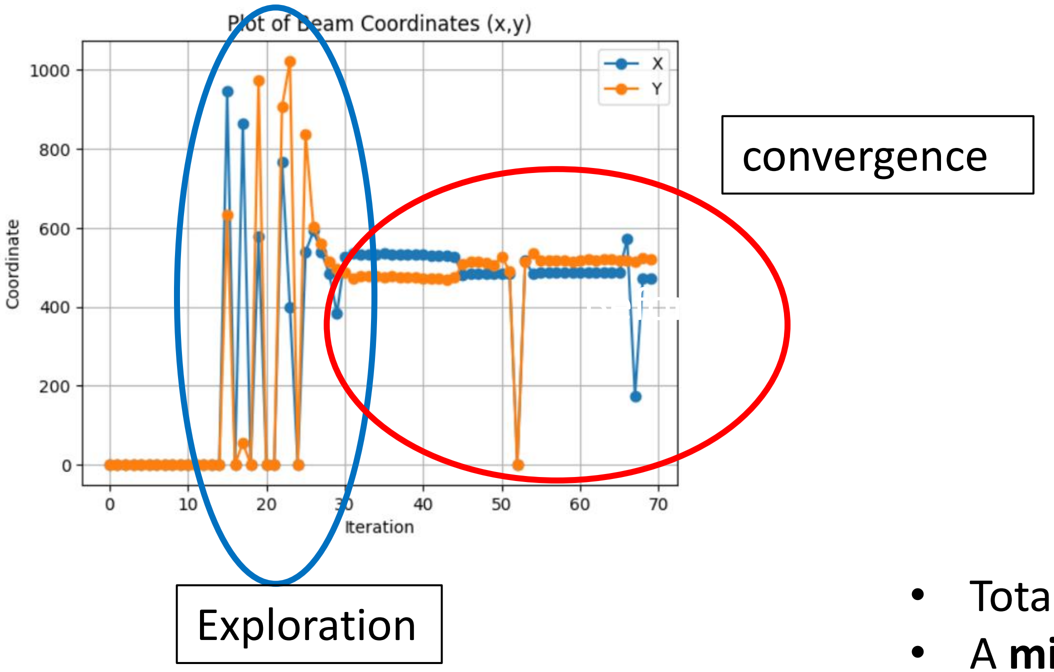


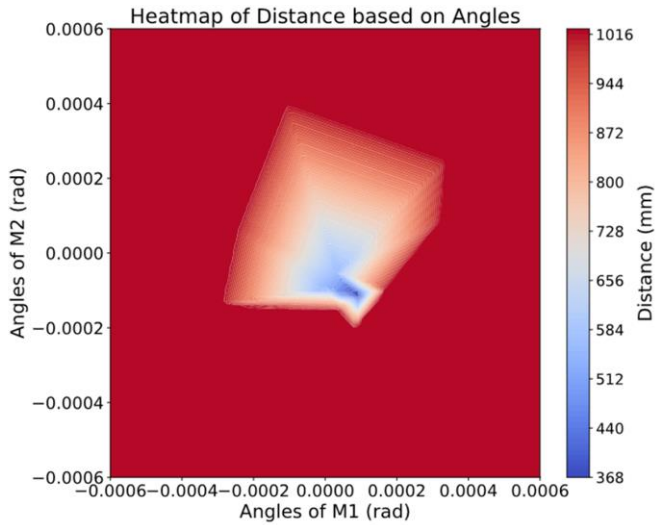


- Successful centering of the beam in the centre of the screen after (BO)
- After iteration 30 both angles **stabilize**

Results

Control of the steering magnets

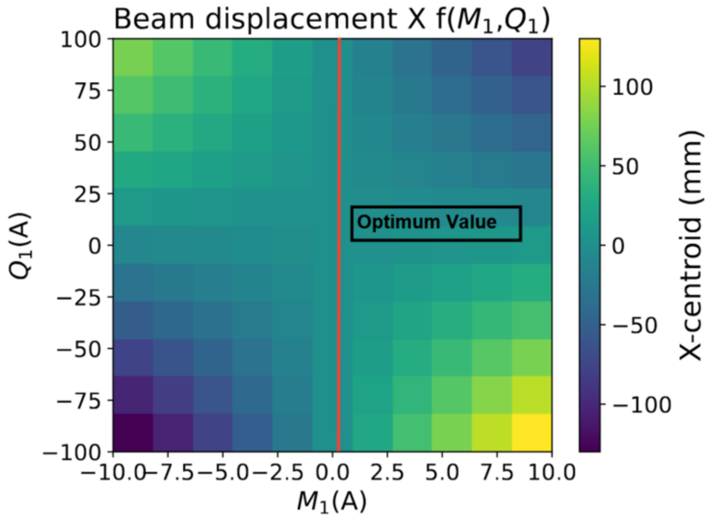


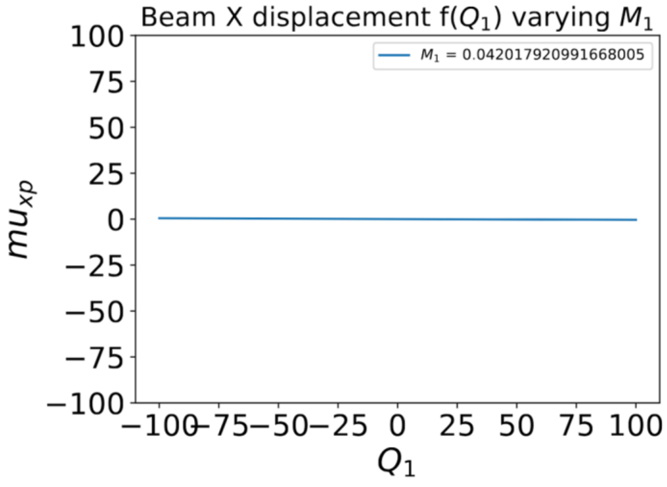


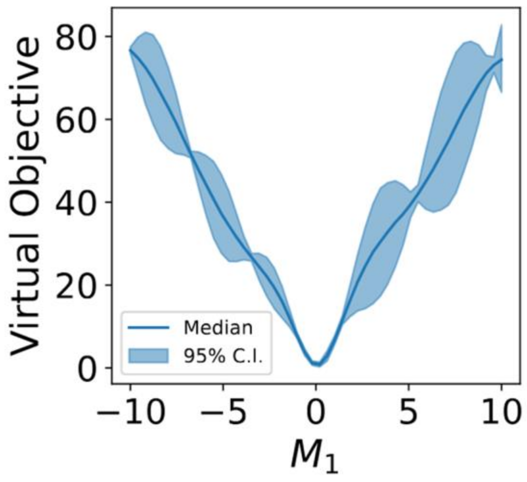
- Total beam displacement as a function of these angular settings
- A **minimum region** appears where the beam approaches the optical axis
- Highlights the **strong sensitivity of the trajectory** to small steering deviations
- Defines a low-displacement region that would serve as an ideal starting point or prior for downstream optimization

Results

BAX to align beam through quadrupole centre (M_1, Q_1)



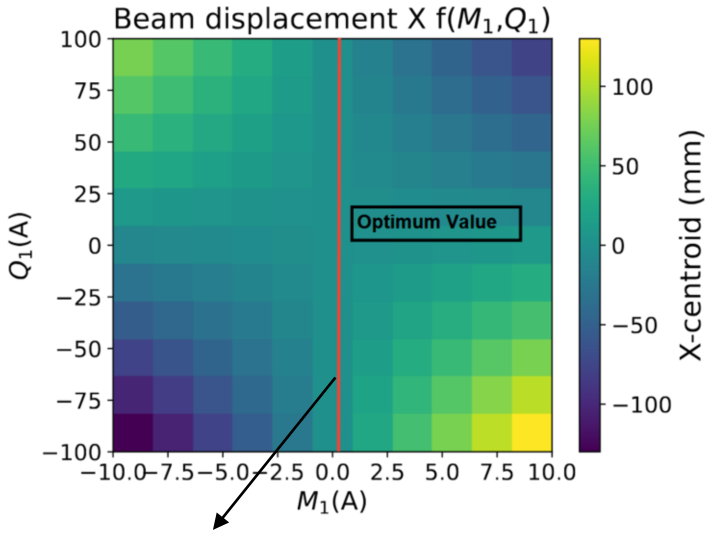


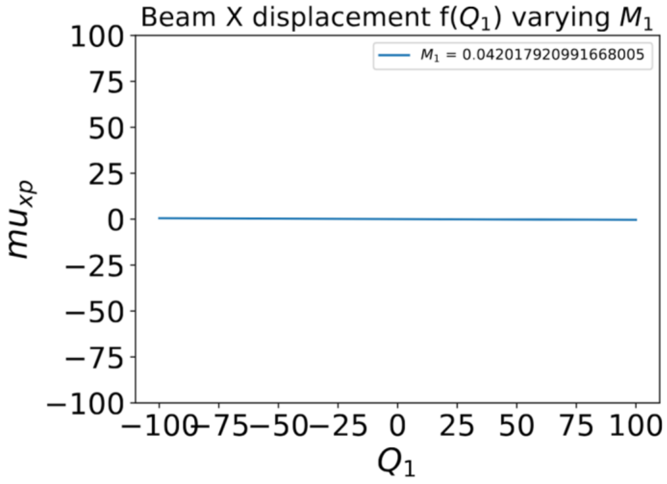


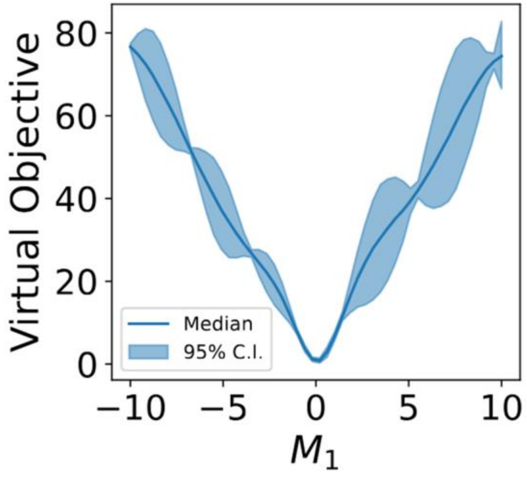
- Need of multi-objective algorithm
- **Minimize a virtual objective** defined as the difference slope in the position of the beam, calculated by varying the quadrupole strength $Q1$

Results

BAX to align beam through quadrupole centre (M_1, Q_1)





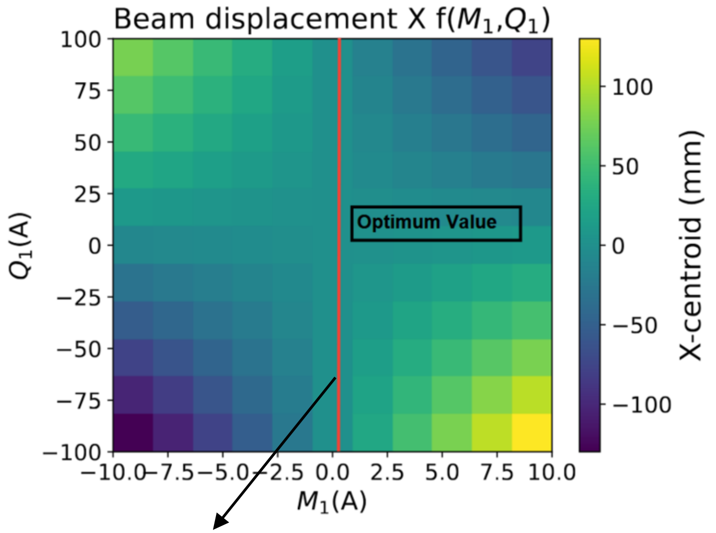


The beam remains minimally displaced across varying quadrupole strengths, indicating ideal alignment.

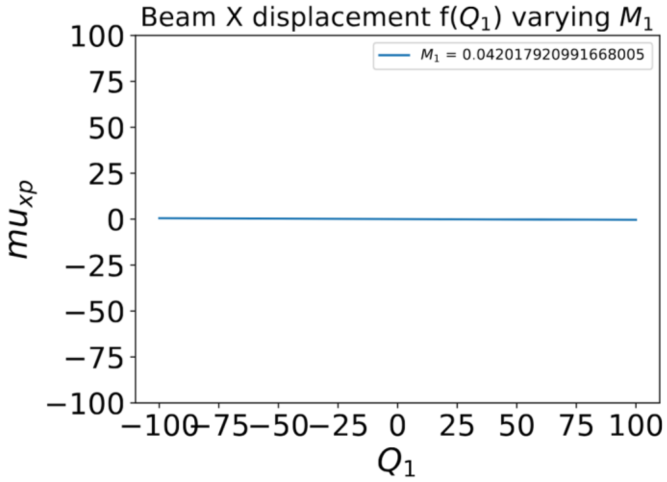
- Need of multi-objective algorithm
- **Minimize a virtual objective** defined as the difference slope in the position of the beam, calculated by varying the quadrupole strength $Q1$

Results

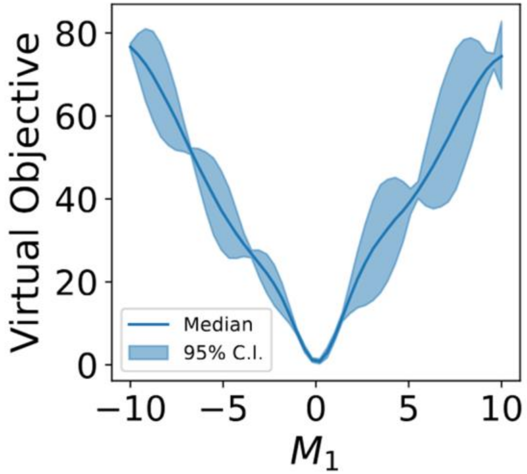
BAX to align beam through quadrupole centre (M_1, Q_1)



The beam remains minimally displaced across varying quadrupole strengths, indicating ideal alignment



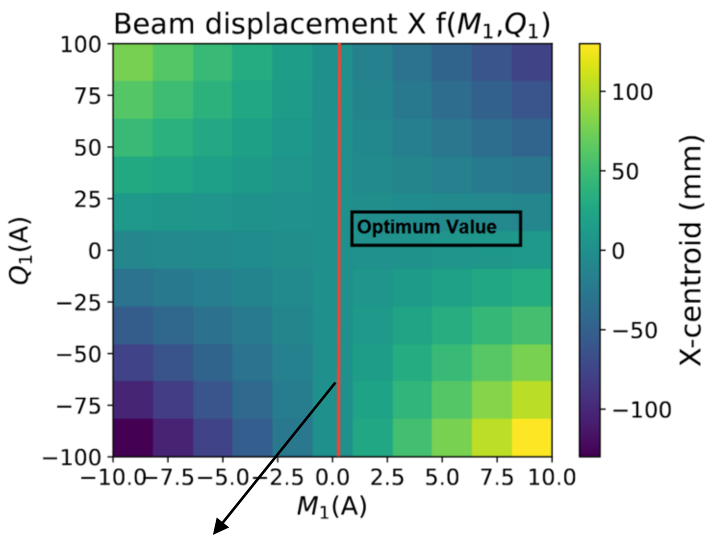
The beam x-displacement as a function of quadrupole strength $Q1$ is near zero, confirming flat response and effective decoupling



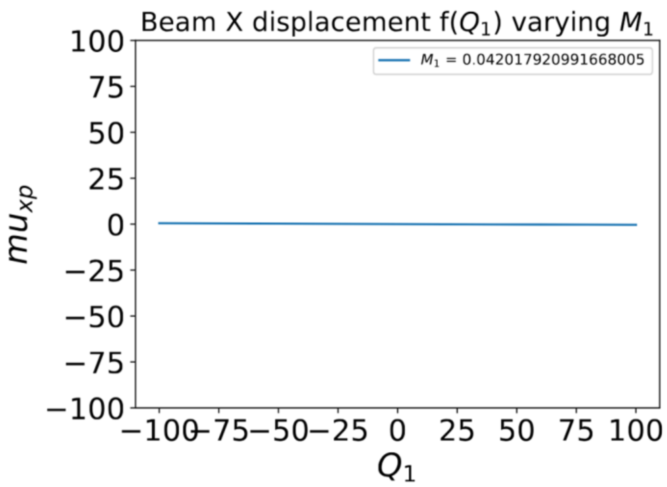
- Need of multi-objective algorithm
- **Minimize a virtual objective** defined as the difference slope in the position of the beam, calculated by varying the quadrupole strength $Q1$

Results

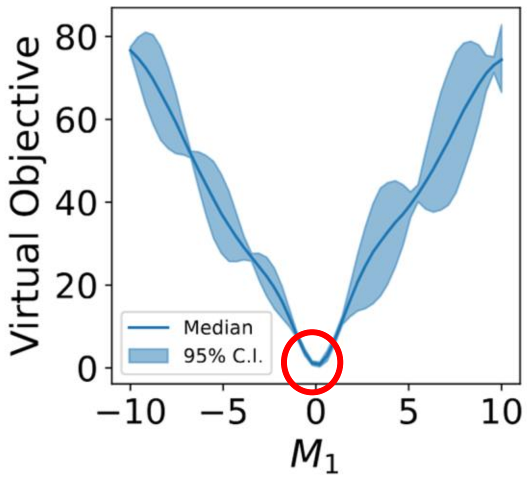
BAX to align beam through quadrupole centre (M_1, Q_1)



The beam remains minimally displaced across varying quadrupole strengths, indicating ideal alignment.



The beam x-displacement as a function of quadrupole strength $Q1$ is near zero, confirming flat response and effective decoupling



BAX algorithm's ability to infer optimal alignment from indirect measurements with high confidence

- Need of multi-objective algorithm
- **Minimize a virtual objective** defined as the difference slope in the position of the beam, calculated by varying the quadrupole strength $Q1$

Conclusion

Now

- Extract data from KARA, injection line beam measurements
- Perform Simulations with Cheetah using (BO) and BAX. Steering magnets, quadrupoles
- Test (BO) and BAX code in KARA facility.

Maybe RL? Open for discussions


Thank you!