Image Inference (Experiment 1)

Preprocessing

```
# Read in the participant data.
data_0 = read_csv(file.path(human_path, "raw_data.csv"), quote="~")

# Read in the MTurk results file.
mturk_results = read_csv(file.path(human_path, "mturk_results.csv"),
                         col_names=TRUE) %>%
  mutate(Answer.surveycode=substr(Answer.surveycode, 3,
                                  length(Answer.surveycode))) %>%
  filter(AssignmentStatus=="Approved")

# Check for invalid survey codes.
invalid = sum(!(mturk_results$Answer.surveycode %in% data_0$unique_id))
if (invalid != 0) { stop("There's an invalid survey code.") }

# Convert the JSON string into JSON.
data_1 = lapply(data_0$results, fromJSON)

# Extract the trial information for each participant and stack them.
age = c()
data_3 = tibble()
for (p in 1:length(data_1)) {
  # Trim the map and add the participant ID back in.
  data_2 = data_1[p][[1]]$trials %>%
    as.data.frame() %>%
    mutate(map=gsub(".png", "", map), unique_id=data_0$unique_id[p],
           wrong_attempts=data_1[p][[1]]$catch_trials$wrong_attempts)

  # Extract and store this participant's age.
  age = c(age, as.integer(data_1[p][[1]]$subject_information$age))

  # Stack the trial information for the current participant.
  data_3 = rbind(data_3, data_2)
}

# Write the preprocessed data.
write_csv(data_3, file.path(human_path, "data.csv"))

# Combine the posterior over goals for each of the maps.
model_1 = tibble()
goal_files = list.files(model_path, pattern="_goal")
for (goal_file in goal_files) {
  # Read in the goal predictions for this map and do some preprocessing.
```

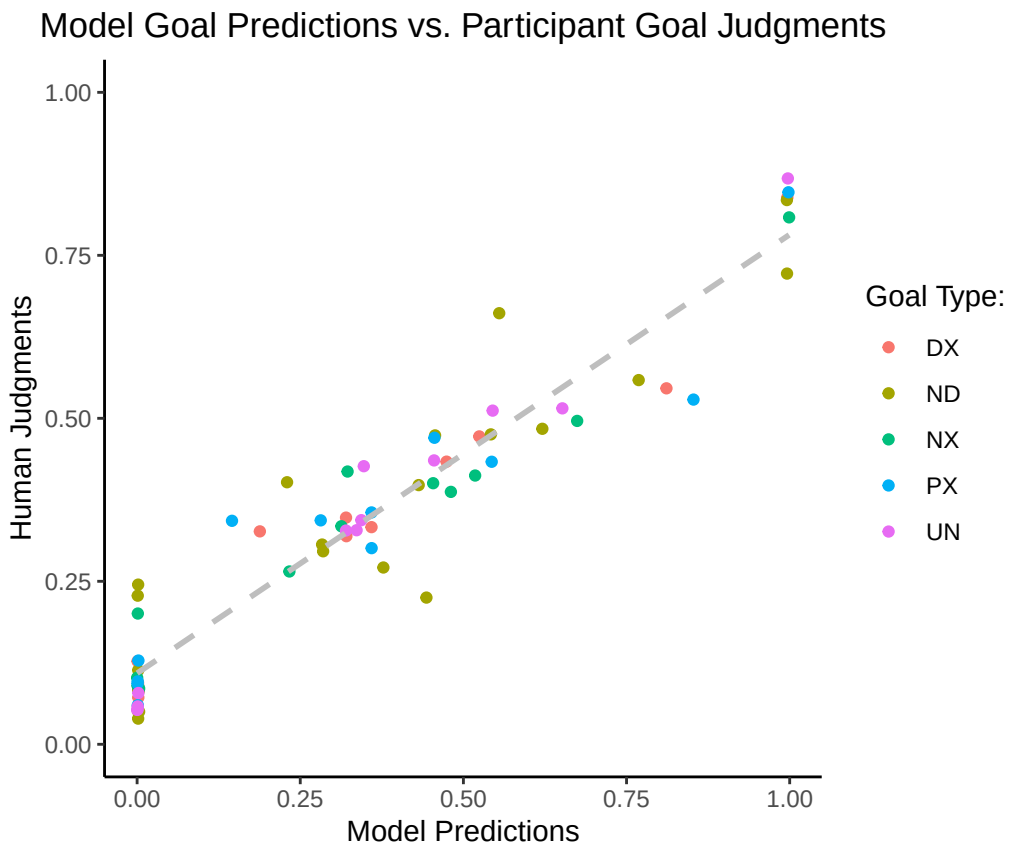
```
  model_0 = read_csv(file.path(model_path, goal_file)) %>%
    mutate(map=gsub("_goals_posterior.csv", "", goal_file),
           goal=ifelse(goal=="Blue", "A",
                       ifelse(goal=="Orange", "B", "C")))

  # Stack these goal predictions.
  model_1 = model_1 %>%
    rbind(model_0)
}

# Write the goal predictions.
write_csv(model_1, file.path(model_path, "goal_predictions.csv"))

# Combine the posterior over entrances for each of the maps.
model_5 = tibble()
state_files = list.files(model_path, pattern="_state")
for (state_file in state_files) {
  # Read in the path predictions.
  model_2 = read_csv(file.path(model_path, state_file))

  # Do some preprocessing.
  model_3 = model_2 %>%
    rownames_to_column("path") %>%
    gather(step, state, names(model_2)[grepl("s_", names(model_2))]) %>%
    separate(step, into=c("temp", "time"), sep="_") %>%
    mutate(path=as.numeric(path)-1, time=as.numeric(time),
           state=as.numeric(state)) %>%
    arrange(path, time) %>%
    mutate(x=state%%map_width+1, y=ceiling(state/map_width)) %>%
    select(-temp, -map_height, -map_width)

  # Extract the the first state from each path (i.e., the entrance).
  model_4 = model_3 %>%
    filter(time==0) %>%
    group_by(state) %>%
    summarize(probability=sum(probability)) %>%
    rename(entrance=state) %>%
    mutate(map=gsub("_states_posterior.csv", "", state_file))

  # Stack these entrance predictions.
  model_5 = model_5 %>%
    rbind(model_4)
}

# Write the entrance predictions.
write_csv(model_5, file.path(model_path, "entrance_predictions.csv"))
```

Goal Inference

Here we generate a scatter plot comparing participant judgments (N=40; M=37.02 years, SD=11.2 years) against our model predictions on the goal inferences.

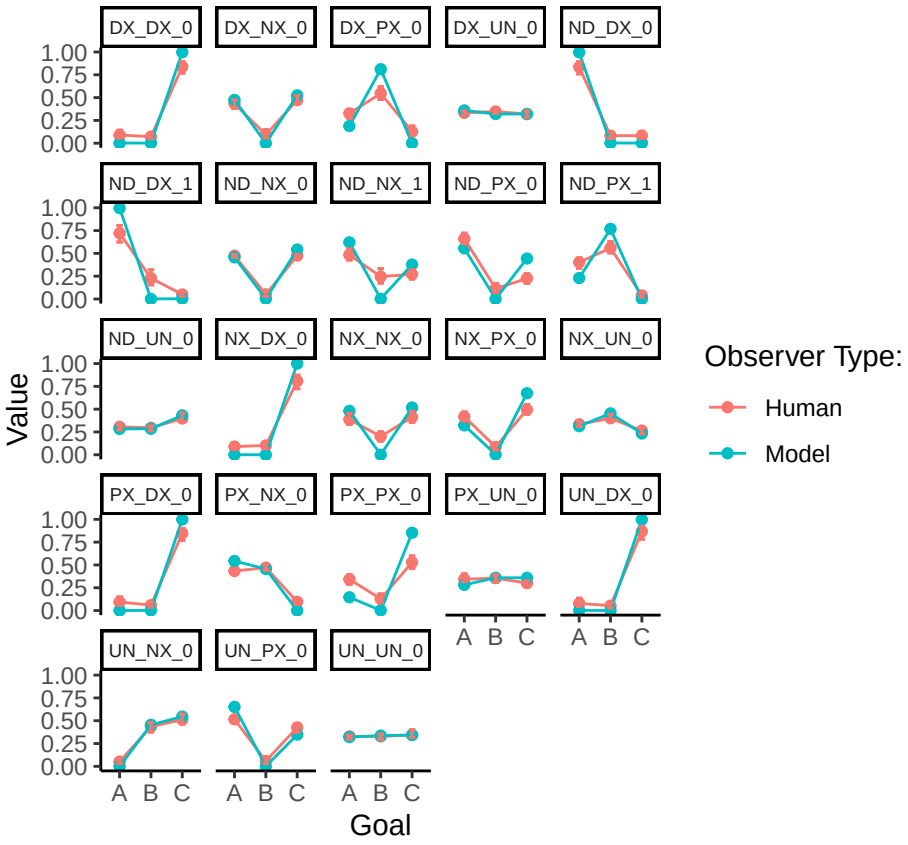
```
# Read in the preprocessed data.
data_3 = read_csv(file.path(human_path, "data.csv"))

# Select and normalize the goal judgments.
data_4 = data_3 %>%
  select(unique_id, map, A, B, C) %>%
  gather(goal, human, A, B, C) %>%
  left_join(do(., summarize(group_by(., unique_id, map),
                            total_human=sum(human)))) %>%
  mutate(human=human/total_human) %>%
  select(-total_human)

# Define the bootstrap function for the bootstrap statistic.
compute_mean = function(data, indices) {
  return(mean(data[indices]))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_mean,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CIs.
set.seed(seed)
ci = data.frame()
for (m in unique(data_4$map)) {
  # Filter the current map.
  data_5 = data_4 %>%
    filter(map==m)

  # Compute the bootstrap for each dependent measure.
  bootstrap_A = compute_bootstrap(filter(data_5, goal=="A")$human)
  bootstrap_B = compute_bootstrap(filter(data_5, goal=="B")$human)
  bootstrap_C = compute_bootstrap(filter(data_5, goal=="C")$human)

  # Store the bootstrapped 95% CIs for this pair.
  ci = rbind(ci, data.frame(map=rep(m, 3),
                            goal=c("A", "B", "C"),
                            lower=c(bootstrap_A[4],
                                    bootstrap_B[4],
                                    bootstrap_C[4]),
                            upper=c(bootstrap_A[5],
                                    bootstrap_B[5],
                                    bootstrap_C[5])))
}

# Read in the goal predictions.
model_6 = read_csv(file.path(model_path, "goal_predictions.csv"))
```

```r
# Perform some basic preprocessing and then z-score the model predictions.
model_7 = model_6 %>%
  rename(model=probability) %>%
  mutate(z_model=scale(model)[,1])

# z-score the participant judgments and then merge them with the bootstrapped
# 95% CIs and the model predictions.
data_6 = data_4 %>%
  group_by(unique_id) %>%
  mutate(z_human=scale(human)[,1]) %>%
  ungroup() %>%
  group_by(map, goal) %>%
  summarize(mean_human=mean(human), mean_z_human=mean(z_human)) %>%
  ungroup() %>%
  left_join(ci) %>%
  left_join(model_7)

# Plot the goal comparison.
plot_0 = data_6 %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Model Goal Predictions vs. Participant Goal Judgments") +
  xlab("Model Predictions") +
  ylab("Human Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Goal Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_0
```



For the goal inferences, the Pearson correlation is r=0.95 (95% CI: 0.92-0.97). Next, we generate goal inference comparisons for each trial.

```
# Plot goal inferences by trial.
plot_1 = data_6 %>%
  gather(type, value, mean_human, model) %>%
  ggplot(aes(x=goal, y=value, group=type)) +
  geom_point(aes(color=type)) +
  geom_line(aes(color=type)) +
  geom_errorbar(aes(ymin=lower, ymax=upper), color="#F8766D", width=0.2) +
  facet_wrap(~map) +
  xlab("Goal") +
  ylab("Value") +
  scale_color_discrete(name="Observer Type:",
                       limits=c("mean_human", "model"),
                       labels=c("Human", "Model")) +
  theme_classic() +
  theme(aspect.ratio=1.0,
        legend.title=element_text(hjust=0.5),
        strip.text=element_text(size=7))
plot_1
```



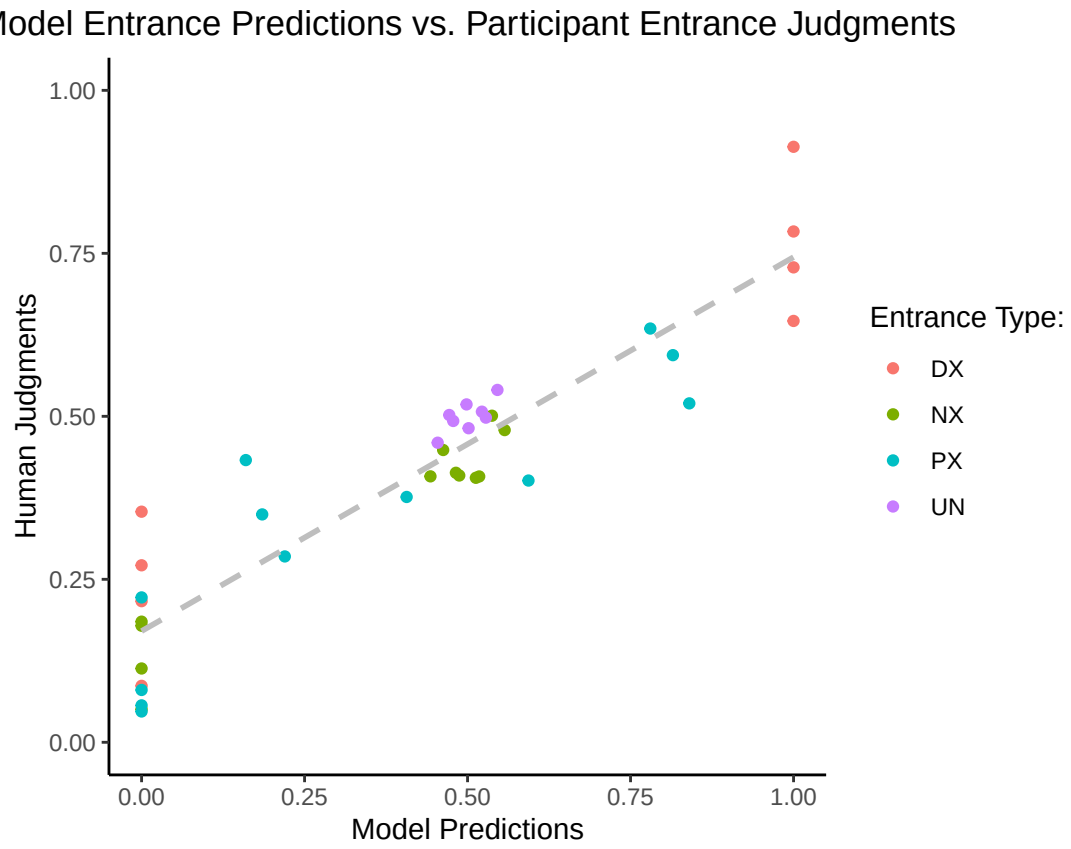
Entrance Inference

Here we generate a scatter plot comparing participant judgments (N=40; M=37.02 years, SD=11.2 years) against our model predictions on the entrance inferences.

```
# Select and normalize the entrance judgments.
data_7 = data_3 %>%
  filter(!grepl("ND", map)) %>%
  select(unique_id, map, `1`, `2`, `3`) %>%
  gather(entrance, human, `1`, `2`, `3`) %>%
  filter(human!=-1) %>%
  left_join(do(., summarize(group_by(., unique_id, map),
                            total_human=sum(human)))) %>%
  mutate(human=human/total_human) %>%
  select(-total_human)

# Define the bootstrap function for the bootstrap statistic.
compute_mean = function(data, indices) {
  return(mean(data[indices]))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
```

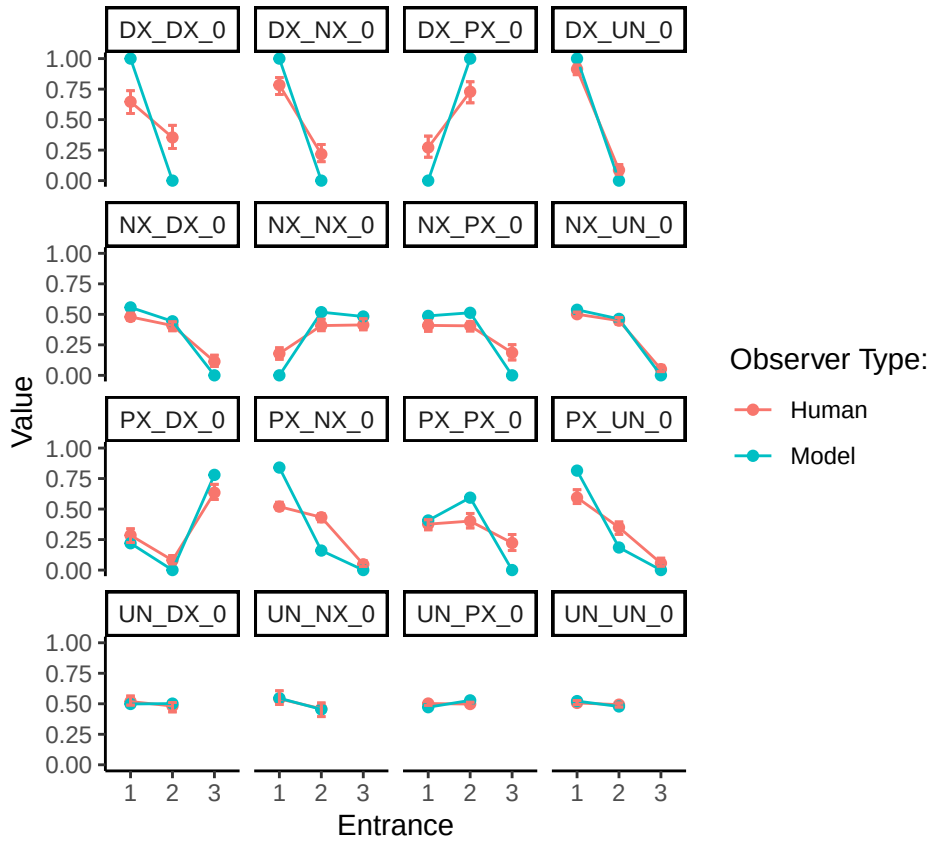
```
  simulations = boot(data=data,
                     statistic=compute_mean,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CIs.
set.seed(seed)
ci = data.frame()
for (m in unique(data_7$map)) {
  # Filter the current map.
  data_8 = data_7 %>%
    filter(map==m)

  # Compute the bootstrap for each dependent measure.
  bootstrap_1 = compute_bootstrap(filter(data_8, entrance=="1")$human)
  bootstrap_2 = compute_bootstrap(filter(data_8, entrance=="2")$human)
  if (length(unique(data_8$entrance)) == 3) {
    bootstrap_3 = compute_bootstrap(filter(data_8, entrance=="3")$human)
  }

  # Store the bootstrapped 95% CIs for this pair.
  if (length(unique(data_8$entrance)) == 3) {
    ci = rbind(ci, data.frame(map=rep(m, 3),
                              entrance=c("1", "2", "3"),
                              lower=c(bootstrap_1[4],
                                      bootstrap_2[4],
                                      bootstrap_3[4]),
                              upper=c(bootstrap_1[5],
                                      bootstrap_2[5],
                                      bootstrap_3[5])))
  }
  else {
    ci = rbind(ci, data.frame(map=rep(m, 2),
                              entrance=c("1", "2"),
                              lower=c(bootstrap_1[4],
                                      bootstrap_2[4]),
                              upper=c(bootstrap_1[5],
                                      bootstrap_2[5])))
  }
}

# Read in the entrance predictions.
model_8 = read_csv(file.path(model_path, "entrance_predictions.csv"))

# Stitch the entrance mapping to the predictions and do some preprocessing.
model_9 = model_8 %>%
  right_join(read_csv(file.path(model_path, "entrance_mapping.csv"))) %>%
  mutate(probability=ifelse(is.na(probability), 0, probability)) %>%
  select(-entrance) %>%
  rename(entrance=number, model=probability) %>%
  mutate(entrance=as.character(entrance), z_model=scale(model)[,1])
```

```
# z-score the participant judgments and then merge them with the bootstrapped
# 95% CIs and the model predictions.
data_9 = data_7 %>%
  group_by(unique_id) %>%
  mutate(z_human=scale(human)[,1]) %>%
  ungroup() %>%
  group_by(map, entrance) %>%
  summarize(mean_human=mean(human), mean_z_human=mean(z_human)) %>%
  ungroup() %>%
  left_join(ci) %>%
  left_join(model_9)

# Plot the entrance comparison.
plot_2 = data_9 %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Model Entrance Predictions vs. Participant Entrance Judgments") +
  xlab("Model Predictions") +
  ylab("Human Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Entrance Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_2
```



For the entrance inferences, the Pearson correlation is r=0.92 (95% CI: 0.86-0.95). Next, we generate entrance inference comparisons for each trial.

```
# Plot entrance inferences by trial.
plot_3 = data_9 %>%
  gather(type, value, mean_human, model) %>%
  ggplot(aes(x=entrance, y=value, group=type)) +
  geom_point(aes(color=type)) +
  geom_line(aes(color=type)) +
  geom_errorbar(aes(ymin=lower, ymax=upper), color="#F8766D", width=0.2) +
  facet_wrap(~map) +
  xlab("Entrance") +
  ylab("Value") +
  scale_color_discrete(name="Observer Type:",
                       limits=c("mean_human", "model"),
                       labels=c("Human", "Model")) +
  theme_classic() +
  theme(aspect.ratio=1.0,
        legend.title=element_text(hjust=0.5))
plot_3
```

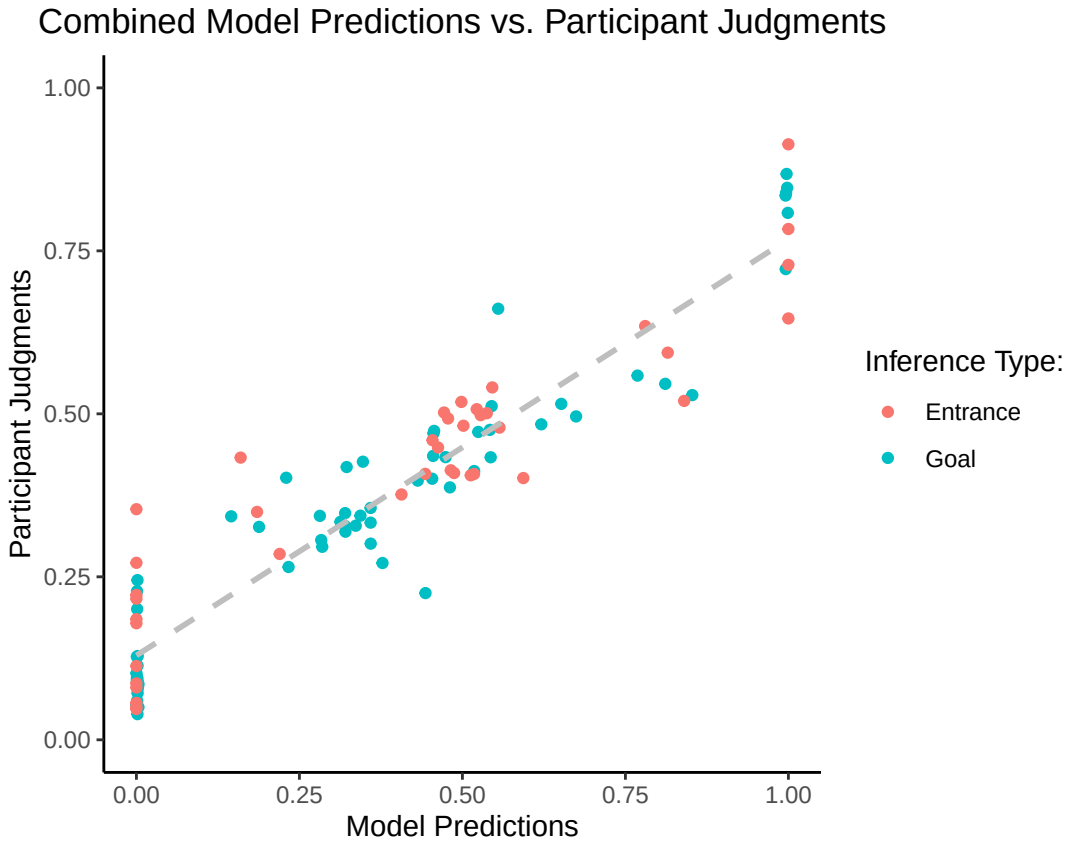


Combined Inference

Finally, we can combine both scatter plots.

```
# Merge the two inferences.
data_10 = data_6 %>%
  mutate(type="goal", inference=goal) %>%
  select(-goal) %>%
  rbind(select(mutate(data_9, type="entrance", inference=entrance), -entrance))

plot_4 = data_10 %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=type)) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Combined Model Predictions vs. Participant Judgments") +
  xlab("Model Predictions") +
  ylab("Participant Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Inference Type:",
                       limits=c("entrance", "goal"),
                       labels=c("Entrance", "Goal")) +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
```

```
        legend.title=element_text(hjust=0.5))
plot_4
```



The combined Pearson correlation is r=0.94 (95% CI: 0.91-0.96).

Alternative Models

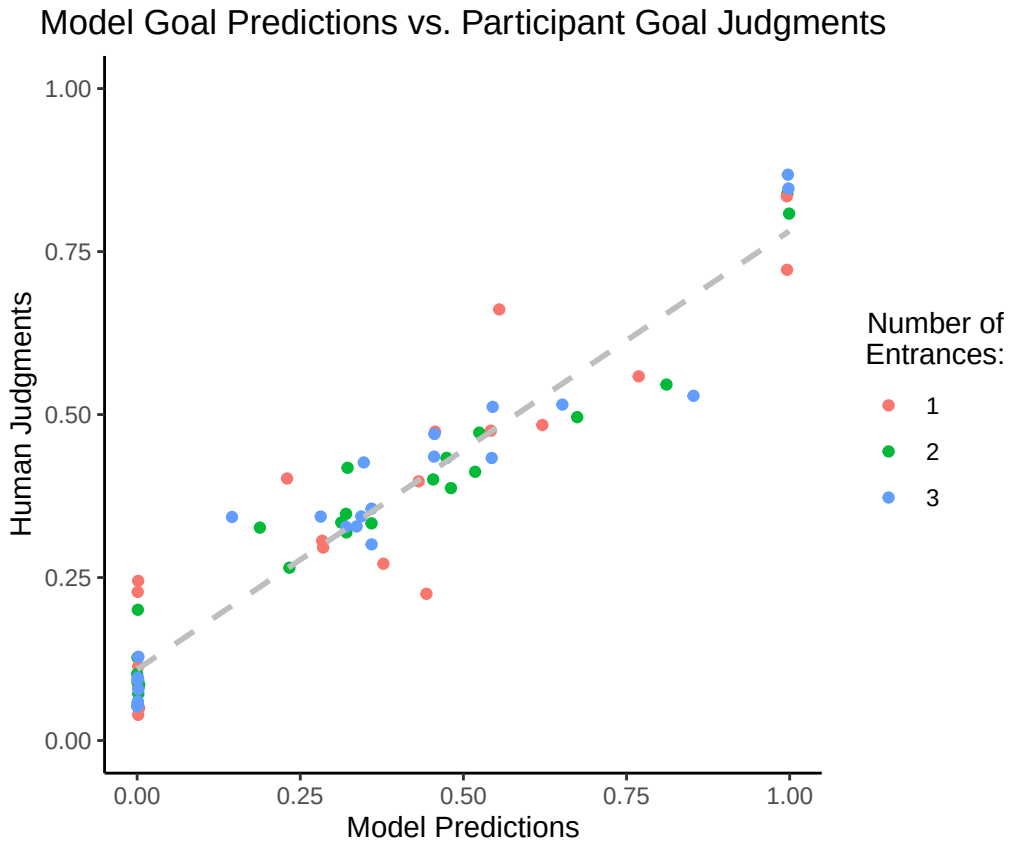
Predicting model error as a function of map complexity

Map complexity as the number of entrances

Now we test if we can predict model error as a function of how many doors there are in each map. First, we generate a scatter plot of the goal inferences that colors each point based on the number of entrances.

```
plot_5 = data_6 %>%
  mutate(num_entrances=ifelse(grepl("ND", substr(map, 1, 2)), 1,
                              ifelse(grepl("DX|NX", substr(map, 1, 2)), 2,
                                     3))) %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=factor(num_entrances))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Model Goal Predictions vs. Participant Goal Judgments") +
  xlab("Model Predictions") +
```

```
  ylab("Human Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Number of\nEntrances:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_5
```



There doesn't seem to be anything here, but let's compute a linear regression that predicts model error on the goal inferences as a function of the number of entrances in each map.

```
data_11 = data_6 %>%
  mutate(model_error=abs(mean_human-model)) %>%
  group_by(map) %>%
  summarize(mean_model_error=mean(model_error)) %>%
  mutate(num_entrances=ifelse(grepl("ND", substr(map, 1, 2)), 1,
                              ifelse(grepl("DX|NX", substr(map, 1, 2)), 2, 3)))
alternative_0_goal_error = lm(formula=mean_model_error~num_entrances,
                              data=data_11)
summary(alternative_0_goal_error)
```

```
##
## Call:
```

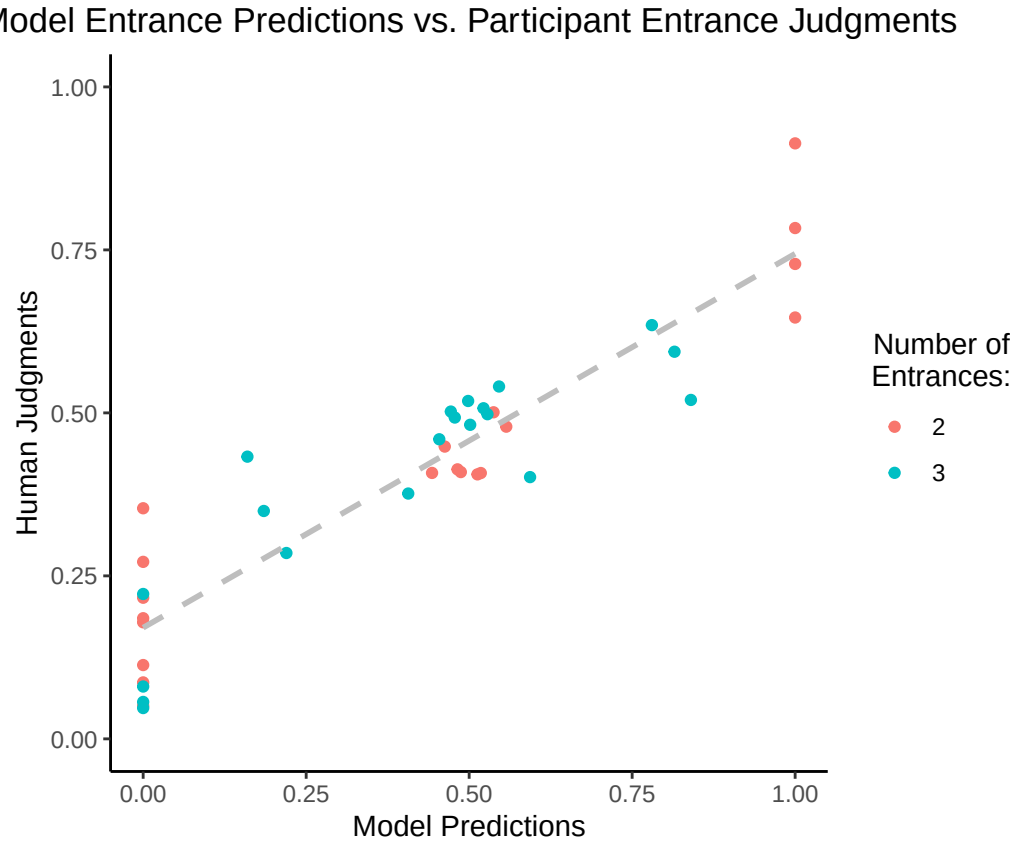
```
## lm(formula = mean_model_error ~ num_entrances, data = data_11)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.092005 -0.042571  0.007373  0.030246  0.135079
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.13149    0.03290   3.997 0.000655 ***
## num_entrances -0.01689    0.01498  -1.128 0.272216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05792 on 21 degrees of freedom
## Multiple R-squared:  0.05709,    Adjusted R-squared:  0.01219
## F-statistic: 1.271 on 1 and 21 DF,  p-value: 0.2722
```

With an R-squared of R^2=0.06, the number of entrances is likely a poor predictor of model error on the goal inferences. Let's compute the correlation between the predictions of the alternative model and the model error to check.

```
# Combine the alternative model predictions with the model error.
data_12 = data_11 %>%
  mutate(alternative=as.vector(fitted(alternative_0_goal_error))) %>%
  select(map, mean_model_error, alternative)

# Define the bootstrap function for the bootstrap statistic.
compute_cor = function(data, indices) {
  return(cor(data$alternative[indices], data$mean_model_error[indices],
             method="pearson"))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CI.
set.seed(seed)
cor_3 = cor(data_12$alternative, data_12$mean_model_error, method="pearson")
cor_3_bootstrap = compute_bootstrap(data_12)
cor_3_ci = data.frame(
  lower=cor_3_bootstrap[4],
  upper=cor_3_bootstrap[5]
)
```

The Pearson correlation of this alternative model is r=0.24 (95% CI: -0.26-0.61). The number of entrances is a poor predictor of model error on the goal inferences. Next, we generate a scatter plot of the entrance inferences that colors each point based on the number of entrances.

```
plot_6 = data_9 %>%
  mutate(num_entrances=ifelse(grepl("ND", substr(map, 1, 2)), 1,
                              ifelse(grepl("DX|NX", substr(map, 1, 2)), 2,
                                     3))) %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=factor(num_entrances))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Model Entrance Predictions vs. Participant Entrance Judgments") +
  xlab("Model Predictions") +
  ylab("Human Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Number of\nEntrances:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_6
```



There also doesn't seem to be anything here, but let's compute another linear regression that attempts to predict model error on the entrance inferences as a function of the number of entrances.

```
data_13 = data_9 %>%
  mutate(model_error=abs(mean_human-model)) %>%
  group_by(map) %>%
  summarize(mean_model_error=mean(model_error)) %>%
```

```
  mutate(num_entrances=ifelse(grepl("ND", substr(map, 1, 2)), 1,
                              ifelse(grepl("DX|NX", substr(map, 1, 2)), 2, 3)))
alternative_0_entrance_error = lm(formula=mean_model_error~num_entrances,
                                  data=data_13)
summary(alternative_0_entrance_error)
```

```
##
## Call:
## lm(formula = mean_model_error ~ num_entrances, data = data_13)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.12620 -0.07061 -0.03873  0.06306  0.19368
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.31098    0.12173   2.555   0.0229 *
## num_entrances -0.07548    0.04775  -1.581   0.1363
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09549 on 14 degrees of freedom
## Multiple R-squared:  0.1514, Adjusted R-squared:  0.09084
## F-statistic: 2.499 on 1 and 14 DF,  p-value: 0.1363
```

With an R^2=0.15, the number of entrances is likely a poor predictor of model error on the entrance inferences. Let's compute the correlation between the predictions of the alternative model and the model error to check.

```
# Combine the alternative model predictions with the model error.
data_14 = data_13 %>%
  mutate(alternative=as.vector(fitted(alternative_0_entrance_error))) %>%
  select(map, mean_model_error, alternative)

# Define the bootstrap function for the bootstrap statistic.
compute_cor = function(data, indices) {
  return(cor(data$alternative[indices], data$mean_model_error[indices],
             method="pearson"))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CI.
set.seed(seed)
cor_4 = cor(data_14$alternative, data_14$mean_model_error, method="pearson")
cor_4_bootstrap = compute_bootstrap(data_14)
```

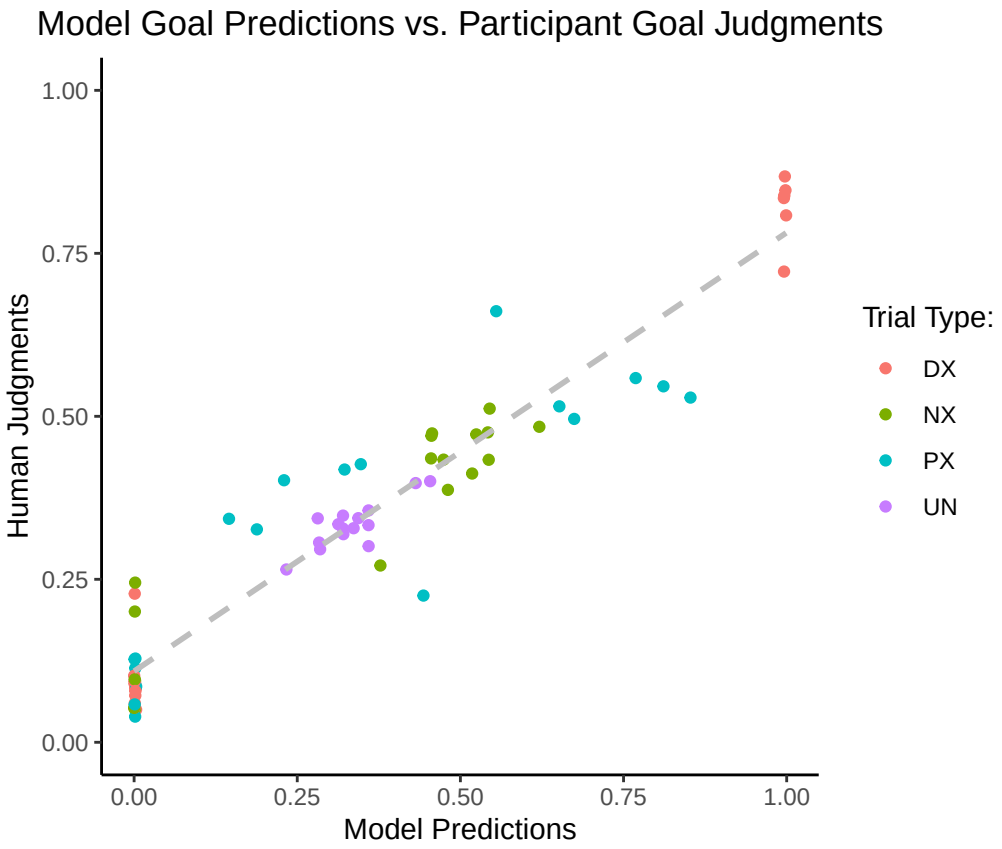
```
cor_4_ci = data.frame(
  lower=cor_4_bootstrap[4],
  upper=cor_4_bootstrap[5]
)
```

The Pearson correlation of this alternative model is r=0.39 (95% CI: -0.13-0.7). The number of entrances is a poor predictor of model error on the entrance inferences.

Map complexity as trial type

Now we test if we can predict model error as a function of the trial types: DX, NX, PX, and UN. First, we generate a scatter plot of the goal inferences that colors each point based on the trial types.

```
plot_7 = data_6 %>%
  mutate(trial_type=substr(map, 4, 5)) %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=factor(trial_type))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Model Goal Predictions vs. Participant Goal Judgments") +
  xlab("Model Predictions") +
  ylab("Human Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Trial Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_7
```



There seems to be higher error in the PX trials, but let's compute a linear regression that predicts model error on the goal inferences as a function of the trial type.

```
data_15 = data_6 %>%
  mutate(model_error=abs(mean_human-model)) %>%
  group_by(map) %>%
  summarize(mean_model_error=mean(model_error)) %>%
  mutate(trial_type=substr(map, 4, 5))
alternative_1_goal_error = lm(formula=mean_model_error~trial_type,
                              data=data_15)
summary(alternative_1_goal_error)
```

```
##
## Call:
## lm(formula = mean_model_error ~ trial_type, data = data_15)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.057087 -0.021233 -0.008019  0.013800  0.077203
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.11824    0.01604   7.371 5.54e-07 ***
## trial_typeNX -0.03325    0.02269  -1.466 0.159128
## trial_typePX  0.02977    0.02269   1.312 0.205140
```

```
## trial_typeUN -0.09360    0.02379  -3.934 0.000891 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0393 on 19 degrees of freedom
## Multiple R-squared:  0.6073, Adjusted R-squared:  0.5453
## F-statistic: 9.793 on 3 and 19 DF,  p-value: 0.000404
```

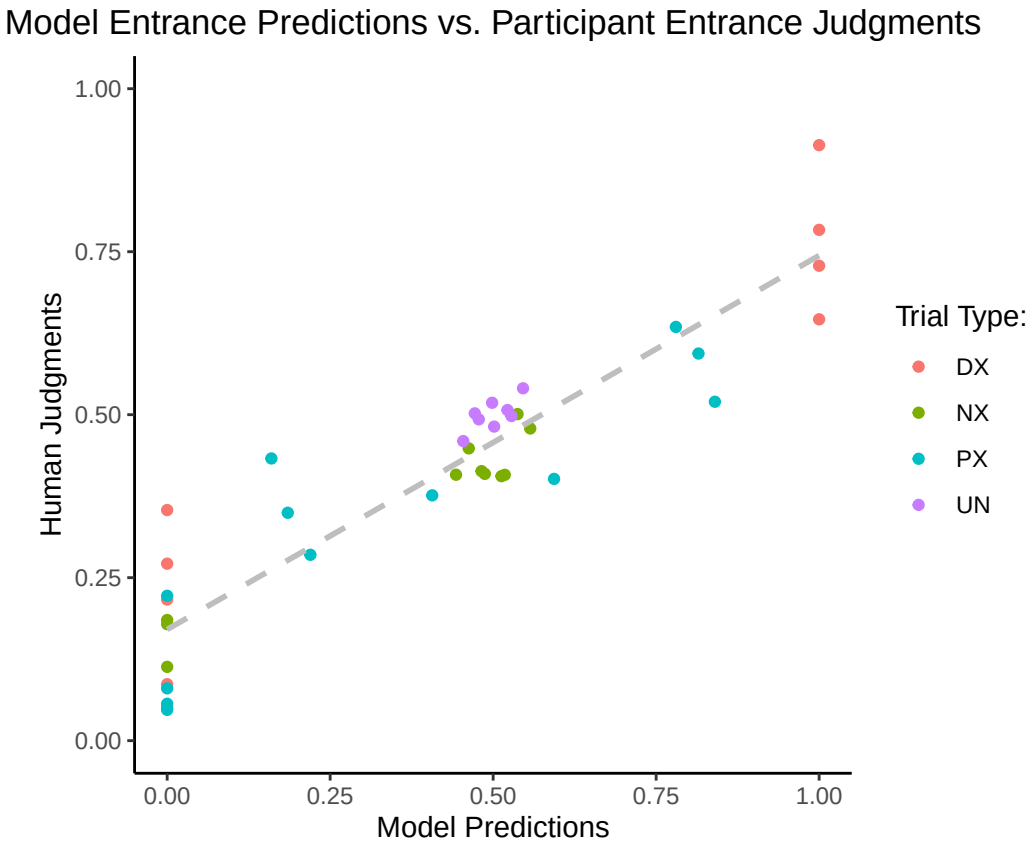
With an R^2=0.61, trial type is likely a good predictor of model error on the goal inferences. Let's compute the correlation between the predictions of the alternative model and the model error to check.

```
# Combine the alternative model predictions with the model error.
data_16 = data_15 %>%
  mutate(alternative=as.vector(fitted(alternative_1_goal_error))) %>%
  select(map, mean_model_error, alternative)

# Define the bootstrap function for the bootstrap statistic.
compute_cor = function(data, indices) {
  return(cor(data$alternative[indices], data$mean_model_error[indices],
             method="pearson"))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CI.
set.seed(seed)
cor_5 = cor(data_16$alternative, data_16$mean_model_error, method="pearson")
cor_5_bootstrap = compute_bootstrap(data_16)
cor_5_ci = data.frame(
  lower=cor_5_bootstrap[4],
  upper=cor_5_bootstrap[5]
)
```

The Pearson correlation of this alternative model is r=0.78 (95% CI: 0.53-0.88). Trial type is a good predictor of model error on the goal inferences. Next, we generate a scatter plot of the entrance inferences that colors each point based on the trial type.

```
plot_8 = data_9 %>%
  mutate(trial_type=substr(map, 1, 2)) %>%
  ggplot(aes(x=model, y=mean_human, label=map)) +
  geom_point(aes(color=factor(trial_type))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Model Entrance Predictions vs. Participant Entrance Judgments") +
  xlab("Model Predictions") +
  ylab("Human Judgments") +
```

```
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Trial Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_8
```



There seems to be clustering among like trials, but let's compute another linear regression that attempts to predict model error on the entrance inferences as a function of the trial type.

```
data_17 = data_9 %>%
  mutate(model_error=abs(mean_human-model)) %>%
  group_by(map) %>%
  summarize(mean_model_error=mean(model_error)) %>%
  mutate(trial_type=substr(map, 1, 2))
alternative_1_entrance_error = lm(formula=mean_model_error~trial_type,
                                  data=data_17)
summary(alternative_1_entrance_error)
```

```
##
## Call:
## lm(formula = mean_model_error ~ trial_type, data = data_17)
##
```

```
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.145477 -0.013271 -0.003039  0.032286  0.121624
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.23208    0.03235   7.175 1.13e-05 ***
## trial_typeNX -0.14410    0.04574  -3.150 0.008373 **
## trial_typePX -0.08052    0.04574  -1.760 0.103799
## trial_typeUN -0.21453    0.04574  -4.690 0.000524 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06469 on 12 degrees of freedom
## Multiple R-squared:  0.6662, Adjusted R-squared:  0.5827
## F-statistic: 7.983 on 3 and 12 DF,  p-value: 0.003427
```

With an R^2=0.67, trial type is likely a good predictor of model error on the entrance inferences. Let's compute the correlation between the predictions of the alternative model and the model error to check.

```
# Combine the alternative model predictions with the model error.
data_18 = data_17 %>%
  mutate(alternative=as.vector(fitted(alternative_1_entrance_error))) %>%
  select(map, mean_model_error, alternative)

# Define the bootstrap function for the bootstrap statistic.
compute_cor = function(data, indices) {
  return(cor(data$alternative[indices], data$mean_model_error[indices],
             method="pearson"))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CI.
set.seed(seed)
cor_6 = cor(data_18$alternative, data_18$mean_model_error, method="pearson")
cor_6_bootstrap = compute_bootstrap(data_18)
cor_6_ci = data.frame(
  lower=cor_6_bootstrap[4],
  upper=cor_6_bootstrap[5]
)
```

The Pearson correlation of this alternative model is r=0.82 (95% CI: 0.41-0.94). Trial type is a good predictor of model error on the entrance inferences.

Predicting goal inference as a function of goal and entrance distance

Now we test whether an alternative model that uses simple distance heuristics performs better than our model. This alternative model consists of a multinomial logistic regression that attempts to predict which goal the agent was going for using (1) the distance between the observation and each goal, (2) the average distance between the observation and each entrance, (3) the number of entrances, and (4) all of their interactions.

```
# Read in the heuristics data for building the alternative model.
data_19 = read_csv("data/experiment_1/alternative/distances.csv")

# Merge the mean participant judgments on the goal inferences with the
# heuristics data.
data_20 = data_6 %>%
  select(map, goal, mean_human) %>%
  spread(goal, mean_human) %>%
  left_join(data_19)

# Compute the average distance between the observation and each entrance and
# the goal that participants rated the highest for each map.
data_22 = tibble()
for (m in 1:length(data_20$map)) {
  # Compute the average entrance distance and max goal for this map.
  data_21 = data_20 %>%
    filter(map==data_20$map[m]) %>%
    mutate(entrance_d=mean(c(`1_d`, `2_d`, `3_d`), na.rm=TRUE),
           max_goal=ifelse(max(A, B, C)==A, 1,
                           ifelse(max(A, B, C)==B, 2, 3)))

  # Stack the data.
  data_22 = data_22 %>%
    rbind(data_21)
}

# Set up the alternative model.
x = model.matrix(max_goal~A_d*B_d*C_d*entrance_d*num_entrances, data_22)[,-1]
y = data_22$max_goal

# Train the alternative model (using LASSO regularization to avoid overfitting)
# and generate predictions using the leave-one-out method.
set.seed(seed)
A = c()
B = c()
C = c()
for (testing_trial in 1:length(data_22$map)) {
  # Set up an array of random regularization parameters.
  lambdas = 10^seq(10, -2, length=100)

  # Partition the data according to the leave-one-out method.
  training_trials = setdiff(1:length(data_22$map), testing_trial)
  x_train = x[training_trials,]
  y_train = y[training_trials]
  x_test = x[testing_trial,]
  y_test = y[testing_trial]
```

```
  # Train the LASSO-regularized model and find the lambda parameter with the
  # best fit.
  lasso_model = glmnet(x_train, y_train, family="multinomial", alpha=1,
                       lambda=lambdas)
  cv_results = cv.glmnet(x_train, y_train, family="multinomial", alpha=1)
  best_lambda = cv_results$lambda.min

  # Generate the prediction of the testing trial.
  predictions = predict(lasso_model, s=best_lambda, newx=t(x_test),
                        family="multinomial")

  # Store the predictions for each goal of this testing trial.
  A = c(A, predictions[1])
  B = c(B, predictions[2])
  C = c(C, predictions[3])
}

# Store the predictions of the alternative model.
alternative = data.frame(
  map=data_22$map,
  A=A,
  B=B,
  C=C
)
```

```
# Stitch the alternative model predictions to the mean participant judgments
# and our model predictions.
data_23 = data_6 %>%
  select(map, goal, mean_human, model) %>%
  left_join(gather(alternative, goal, alternative, A, B, C))

# Define the bootstrap function for the bootstrap statistic.
compute_cor = function(data, indices) {
  return(cor(data$alternative[indices], data$mean_human[indices],
             method="pearson"))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CI.
set.seed(seed)
cor_7 = cor(data_23$alternative, data_23$mean_human, method="pearson")
cor_7_bootstrap = compute_bootstrap(data_23)
cor_7_ci = data.frame(
  lower=cor_7_bootstrap[4],
```

```
  upper=cor_7_bootstrap[5]
)

# Define the bootstrap function for the bootstrap statistic.
compute_cor_diff = function(data, indices) {
  cor_alternative = cor(data$alternative[indices], data$mean_human[indices],
                        method="pearson")
  cor_model = cor(data$model[indices], data$mean_human[indices],
                  method="pearson")
  return(cor_alternative-cor_model)
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_cor_diff,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the correlation difference and the bootstrapped 95% CI.
set.seed(seed)
cor_8 = abs(cor_2-cor_7)
cor_8_bootstrap = compute_bootstrap(data_23)
cor_8_ci = data.frame(
  lower=cor_8_bootstrap[4],
  upper=cor_8_bootstrap[5]
)
```

The Pearson correlation of the alternative model is r=0.49 (95% CI: 0.3-0.63). The correlation difference between the alternative model and our model is Δr=0.46 (95% CI: 0.33-0.65).

Visualizations

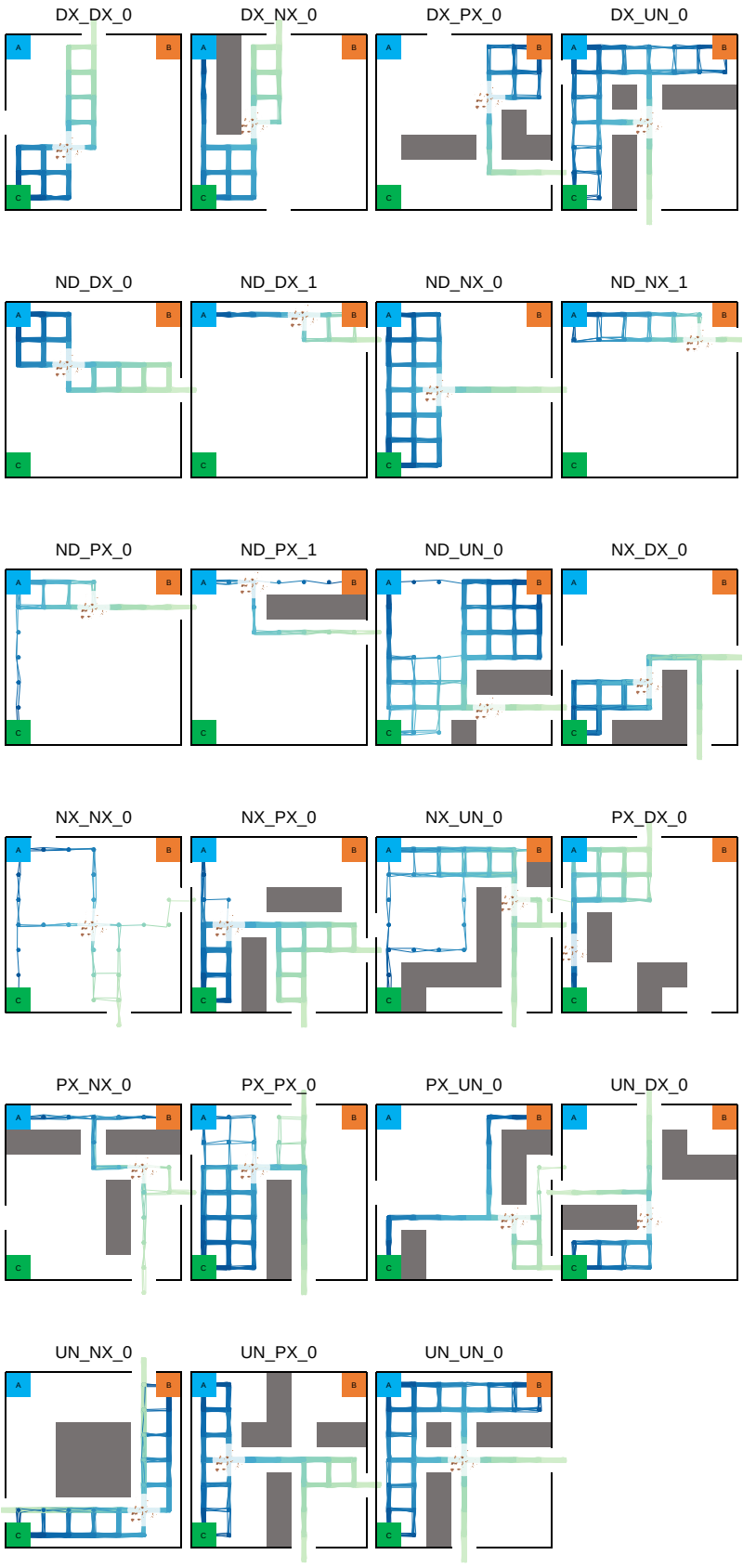
Here we plot the visualizations from the model.

```
# Set the visualization threshold.
threshold = 0.001

# Read in the full set of state inferences.
data_25 = tibble()
for (file in list.files(model_path, pattern="states")) {
  # Read in the state inferences for this map.
  data_24 = read_csv(file.path(model_path, file)) %>%
    mutate(map=gsub("_states_posterior.csv", "", file))

  # Concatenate these state inferences.
  data_25 = data_25 %>%
    bind_rows(data_24)
}
```

```
# Restructure the state inferences data.
data_26 = data_25 %>%
  filter(probability>=threshold) %>%
  rownames_to_column("id") %>%
  gather(step, state,
         sapply(c(0:33), function(x) { return(paste("s_", x, sep="")) })) %>%
  separate(step, into=c("discard", "time"), sep=2) %>%
  select(-discard) %>%
  mutate(id=as.numeric(id), time=as.numeric(time)) %>%
  arrange(id, time) %>%
  mutate(x=state%%map_width+1,
         y=ceiling(state/map_width))

# Trim the states after the goal has been reached since we're only considering
# "entering" states.
data_29 = tibble()
  for (i in unique(data_26$id)) {
    # Filter the current path.
    data_27 = data_26 %>%
      filter(id==i)

    # Trim off any NAs, compute the length of the path with only
    # "entering" states, then trim the path accordingly.
    cutoff = length(na.omit(data_27$state)) / 2
    data_28 = data_27 %>%
      filter(time<cutoff)

    # Concatenate the current path.
    data_29 = data_29 %>%
      rbind(data_28)
  }

# Import the coordinates of the observations, inner walls, and outer walls.
source("stimuli/experiment_1/map_specifications.R")

# Define the coordinates of the goals.
goals = data.frame(
  x_1 = c(2.5, 8.5, 2.5),
  x_2 = c(3.5, 9.5, 3.5),
  y_1 = c(2.5, 2.5, 8.5),
  y_2 = c(3.5, 3.5, 9.5),
  label = c("A", "B", "C")
)

# Define the coordinate bounds.
map_height = 11
map_width = 11

# Set up the general color palette that we're going to visualize with.
palette = colorRampPalette(brewer.pal(9, "GnBu"))

# Generate the visualizations of the participant-generated paths for each map.
plot_10 = list()
```

```
plot_11 = list()
for (trial_num in 1:length(unique(data_29$map))) {
  # Extract the coordinates of the observation for this map.
  crumbs = data.frame(
    x=observations[[trial_num]][1],
    y=observations[[trial_num]][2],
    filename="stimuli/experiment_1/crumbs.png"
  )

  # Compute the specific color palette based on the longest path for this map.
  max_path_length = max(filter(data_29, map==maps[trial_num])$time) + 1
  num_colors = round(max_path_length*1.4)
  color_gradient = palette(num_colors)[(num_colors-max_path_length):num_colors]

  # Generate the visualization for this map.
  plot_9 = data_29 %>%
    filter(map==maps[trial_num]) %>%
    mutate(max_time=max(time)) %>%
    ggplot() +
    geom_path(aes(x=x, y=y, group=id, color=time),
              position=position_jitter(height=0.1, width=0.1)) +
    geom_point(aes(x=x, y=y, group=id, color=time)) +
    scale_x_continuous(name="", limits=c(0, map_width+1)) +
    scale_y_reverse(name="", limits=c(map_height+1, 1)) +
    coord_fixed() +
    theme_void() +
    theme(legend.position="none") +
    geom_point(data=crumbs, aes(x=x, y=y), color="white", alpha=0.8, size=17) +
    geom_image(data=crumbs, aes(x=x, y=y, image=filename), size=0.1) +
    geom_rect(data=goals,
              aes(xmin=x_1, xmax=x_2, ymin=y_1, ymax=y_2, fill=label),
              alpha=1.0) +
    geom_text(data=goals, aes(label=label, x=(x_1+x_2)/2, y=(y_1+y_2)/2),
              size=9*(5/14)) +
    scale_color_gradientn(colors=color_gradient,
                          limits=c(0, max_path_length-1)) +
    scale_fill_manual(values=c("#00AfEf", "#ED7D31", "#00B050", "#767171"))

  # Add the walls to the visualization.
  for (wall in walls[trial_num][[1]]) {
    plot_9 = plot_9 + wall
  }

  # Add the border to the visualization.
  for (segment in segments[trial_num][[1]]) {
    plot_9 = plot_9 + segment
  }

  # Store the plot separately before adding a title and adjusting the margins.
  plot_10[[trial_num]] = plot_9

  # Add a title to the visualization, adjust the margins, and store it.
  plot_11[[trial_num]] = plot_9 +
```

```
    ggtitle(maps[trial_num]) +
    theme(plot.title=element_text(size=20, hjust=0.5, vjust=-10.0),
          plot.margin=unit(c(-2, -3, -2, -3), "cm"))
}

# Plot the visualizations in a grid.
plot_12 = plot_grid(plot_11[[1]], plot_11[[2]], plot_11[[3]], plot_11[[4]],
                    plot_11[[5]], plot_11[[6]], plot_11[[7]], plot_11[[8]],
                    plot_11[[9]], plot_11[[10]], plot_11[[11]], plot_11[[12]],
                    plot_11[[13]], plot_11[[14]], plot_11[[15]], plot_11[[16]],
                    plot_11[[17]], plot_11[[18]], plot_11[[19]], plot_11[[20]],
                    plot_11[[21]], plot_11[[22]], plot_11[[23]],
                    ncol=4)
plot_12
```


DX_DX_0
DX_NX_0
DX_PX_0
DX_UN_0
ND_DX_0
ND_DX_1
ND_NX_0
ND_NX_1
ND_PX_0
ND_PX_1
ND_UN_0
NX_DX_0
NX_NX_0
NX_PX_0
NX_UN_0
PX_DX_0
PX_NX_0
PX_PX_0
PX_UN_0
UN_DX_0
UN_NX_0
UN_PX_0
UN_UN_0

Saving the visualizations

Here we save the visualizations (excluding the title and margin adjustments).

```
# Save the visualizations.
for (trial_num in 1:length(unique(data_29$map))) {
  # Stitch the path for this visualization.
  data_path = paste("data/experiment_1/model/visualizations/Manhattan/",
                    maps[trial_num], sep="")

  # Save this visualization.
  ggsave(paste(data_path, ".png", sep=""), plot_10[[trial_num]], device="png",
         height=4.5, width=7.3)
  ggsave(paste(data_path, ".pdf", sep=""), plot_10[[trial_num]], device="pdf",
         height=4.5, width=7.3)
}
```